UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tadej Mittoni

Iskanje po zbirkah ljudske glasbe na podlagi mrmranja

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2016

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tadej Mittoni

Iskanje po zbirkah ljudske glasbe na podlagi mrmranja

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTORICA: viš. pred. dr. Alenka Kavčič
SOMENTOR: doc. dr. Matija Marolt

Ljubljana, 2016

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V okviru diplomske naloge razvijte sistem za iskanje po glasbenih zbirkah na podlagi mrmranja. Identificirajte najprimernejši algoritem za izločanje zapete melodije iz glasbenega posnetka ter algoritem za približno iskanje po zbirkah. Oba preizkusite na zbirkah ljudske glasbe, pri tem izdelajte ustrezno evalvacijsko množico in poiščite optimalne parametre za delovanje sistema.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Tadej Mittoni sem avtor diplomskega dela z naslovom:

Iskanje po zbirkah ljudske glasbe na podlagi mrmranja

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Alenke Kavčič in somentorstvom doc. dr. Matije Marolta,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 10. februarja 2016 Podpis avtorja:

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
QBH	Query by Humming	Poizvedba z mrmranjem
SMBGT	Subsequence Matching with Bounded Gaps and Tolerances	Ujemanje podvzorcev s pomočjo omejevanja dolžine vmesnih vrzeli ter nastavljivo toleranco
SMGT	Subsequence Matching with Gaps-Range-Tolerances	Ujemanje podvzorcev s pomočjo vrzeli, območja in tolerance
DTW	Dynamic Time Warping	Dinamično ukrivljanje časa
PDA	Pitch Detection Algorithms	Algoritmi za detekcijo višine tona
HMM	Hidden Markov Model	Skriti Modeli Markova
DP	Dynamic Programming	Dinamično programiranje
IOI	Inter-Onset Interval	Razlika v času med začetkoma dveh sosednjih not
IOIR	Inter-Onset Interval Ratio	Razmerje razlike v času med začetkoma dveh sosednjih not
IDE	Integrated Development Environment	Razvojno okolje
PCM	Pulse Code Modulation	Pulzno kodna modulacija

Povzetek

Glavna naloga sistemov QBH je identifikacija vhodnemu zamrmranemu vzorcu najbolj podobnih glasbenih del v podatkovni bazi. Postopek se začne z zajemom zamrmrane melodije na strani uporabnika, nadaljuje s transkripcijo zamrmranega vzorca, nato sistem s pomočjo algoritmov za iskanje podobnosti najde določeno število najbolj podobnih glasbenih del v podatkovni bazi, kar je sporočeno nazaj uporabniku. Cilj diplomskega dela je preučitev že ponujenih sistemov QBH ter pozneje implementacija najoptimalnejšega za uporabo v spletni aplikaciji EtnoFletno. Za fazo transkripcije zamrmranega vzorca sta bila preizkušena algoritma YIN ter njegova izboljšana različica pYIN. Preizkušeni algoritmi za iskanje podobnosti so bili DTW, razdalja urejanja, Spring ter SMGT in SMBGT. Zaradi cilja po optimizaciji tako transkripcijskih kot tudi iskalnih algoritmov so bili vsi poskusi izvedeni s podatkovno bazo, ki je vsebovala slovenske ljudske pesmi. Končni rezultati so pokazali, da je najoptimalnejši sistem QBH sestavljen iz transkripcijskega algoritma pYIN ter iskalnega algoritma SMBGT, ki lahko glede na pevske sposobnosti uporabnika preklaplja med dvema sklopoma parametrov.

Ključne besede: poizvedba z mrmranjem, transkripcija zamrmranega vzorca, EtnoFletno, slovenske ljudske pesmi, algoritmi za iskanje najbolj podobnih glasbenih del, algoritmi za detekcijo višine tona, dinamično programiranje, verjetnostni YIN, ujemanje podvzorca

Abstract

QBH systems are designed to identify the most similar songs in database using hummed query. The process begins by capturing a hummed query on the user side, continues with its transcription using pitch detection algorithms and ends with user being presented the results of search algorithms. The goal of this thesis was to examine existing QBH systems in order to find the most optimal one for use in web application EtnoFletno, which was followed by its implementation. Algorithms YIN and probabilistic YIN were considered and tested for query transcription phase of the target system. Several search algorithms were implemented and tested as well, including DTW, Edit Distance, Spring, SMGT and SMBGT. Transcription and search algorithms had to be optimized for usage in EtnoFletno, hence the testing database contained Slovenian folk songs. Final results show that the transcription algorithm probabilistic YIN was better than its predecessor YIN and algorithm SMBGT outperformed all other search algorithms. It is also shown in the results that algorithm SMBGTs parameters should be used in two different predefined ways considering users singing skills.

Keywords: query by humming, transcription of hummed query, EtnoFletno, Slovenian folk songs, search algorithms, pitch detection algorithms, dynamical programming, probabilistic YIN, subsequence matching

Poglavje 1. Uvod

Glasba je univerzalen jezik. Razvoj glasbene industrije že od nekdaj diktirajo zahteve ljudi ter odkritja s področja znanosti in tehnologije. S hitrim načinom življenja, kot ga imamo zdaj, diktiramo tudi vse večjo zahtevo po hitrosti ter prilagodljivosti iskalnikov. Precej očitno je, da je razvoj popolnoma univerzalnega iskalnika, ki bi upošteval vse uporabnikove zahteve, skoraj nemogoč. Ponujenih nam je bilo že veliko različnih metod za iskanje po ogromnih glasbenih podatkovnih bazah v upanju, da bi nam pomagale pri iskanju najljubšega glasbenega dela. V grobem jih lahko delimo na sledeče [1]:

- Iskanje s pomočjo meta podatkov (biografija izvajalca, napisana mnenja o specifičnem glasbenem delu, datumi koncertov itd.).
- Iskanje s pomočjo dela besedila, ki se nahaja v glasbenem delu.
- Iskanje s pomočjo priporočene podobne glasbe (Pandora, Last.fm, Grooveshark itd.).
- Iskanje s pomočjo posnetka izseka glasbenega dela (Shazam, SoundHound, Musipedia).

Eden bolj priljubljenih iskalnikov glasbe je Shazam. Slednji za identifikacijo celotnega glasbenega dela potrebuje posnetek izseka tega dela. Oseba, ki uporablja iskalnik, tako po navadi priskrbi nekaj sekundni posnetek želenega glasbenega dela. To stori s pomočjo za to razvite mobilne aplikacije za zajem zvoka s pomočjo mikrofona na napravi. Svoje izvorne kode Shazam sicer ne deli z javnostjo, so pa opisali osnovni algoritem, ki jim pomaga z glasbenega posnetka pridobiti vse potrebne informacije. Imajo izjemno veliko podatkovno bazo pesmi različnih žanrov, iz katerih so s pomočjo analize izluščili višine zaigranih tonov, unikatne harmonije in druge akustične lastnosti. Vsako glasbeno delo je nato predstavljeno kot graf frekvence v odvisnosti od časa, ki se imenuje spektrogram. Ravno tako s pomočjo analize pridobijo akustične lastnosti kratkega prejetega posnetka, ki ga nato primerjajo z vsemi glasbenimi deli v podatkovni bazi. Tako je identificirano najbolj podobno glasbeno delo, katerega izvajalec in naslov sta nato posredovana nazaj uporabniku aplikacije.

Najpogosteje so iskalniki glasbe, ki potrebujejo za uspešno identifikacijo skladbe posnetek izseka, uporabljeni v situaciji, ko uporabnik posluša radio in si želi trenutno predvajano neznano skladbo slišati še pozneje. V takšnem primeru Shazam in podobni iskalniki večinoma v prvem poskusu pravilno identificirajo glasbeni posnetek. Obstaja še drugačen scenarij, za katerega pa takšni iskalniki niso primerni, saj je melodija ujeta v naših mislih. Takrat se ne moremo spomniti niti besedila, niti avtorja, lahko pa jo zamrmramo. Človeški možgani si namreč veliko hitreje zapomnijo melodijo pesmi kot njene meta podatke. Ljudje z boljšim posluhom za glasbo so sposobni bolje zamrmrati melodijo, od tistih s slabšim, pri vseh pa lahko nastanejo napake, ki negativno vplivajo na uspešnost klasičnih iskalnikov. Napake, ki nastanejo pri mrmranju melodije, so na primer zgrešena začetna intonacija, napačen tempo, melodija je lahko zapeta v drugem glasbenem ključu ali pa ji je lahko celo

dodana ali izvzeta kakšna vmesna nota. Zaradi kompleksnosti takšne poizvedbe so bili razviti Sistemi iskanja z mrmranjem (angl. Query by Humming ali kasneje QBH).

Sistem QBH predpostavi, da je melodija glasbenega dela njena najpomembnejša lastnost, zato od uporabnika ne zahteva posnetka izseka, temveč le zamrmrano glavno melodijo tega dela. Takšen sistem je z vidika uporabnika preprost za uporabo. V večini primerov je treba le pritisniti gumb za začetek snemanja, zamrmrati melodijo ter nato pritisniti še gumb, ki pošlje poizvedbo. Postopek identifikacije na splošno sestavljajo štirje deli:

- zajem zamrmrane melodije na uporabnikovi strani,
- transkripcija zamrmrane melodije na strani strežnika,
- iskanje najbolj podobnih glasbenih del v podatkovni bazi na strani strežnika in
- izpis nekaj najbolj podobnih glasbenih del zamrmrani melodiji na uporabnikovi strani.

Ponujenih je bilo že kar nekaj različnih pristopov k implementaciji uspešnega sistema QBH. Obstaja implementacija sistema QBH, ki se prilagodi trenutnemu uporabniku [2]. To stori z nekaj preizkusnimi poizvedbami, ki jih uporabnik glede na želene rezultate oceni. Sistem nato s pomočjo genetskega algoritma določi, kakšen tip napak uporabnik naredi in iskanje temu primerno prilagodi. Vse to mu omogoči učni sistem segmentacije not.

Predlagan je tudi sistem, ki za predstavitev frekvenc vhodnega posnetka uporabi Angleško abecedo [3]. Vsak simbol abecede predstavlja koš, v katerem so izračunane frekvence, ki spadajo v določen frekvenčni interval. Zaporedje simbolov se nato s pomočjo algoritma razdalje urejanja [4] med seboj primerja.

Med ponujenimi je tudi sistem, ki temelji na informacijah, pridobljenih iz not [5]. Glavna algoritma v omenjenem sistemu sta NLS[1] ter NRA[2], ki dokazano delujeta izjemno hitro. Sistem, ki kombinira ta dva algoritma, lahko dokazano doseže dobro razmerje med časovno zahtevnostjo in natančnostjo rezultatov.

Hum-a-song [6] je sistem QBH, ki je tako preprost kot tudi učinkovit. V prvi fazi transkripcije zamrmrane melodije sistem uporablja plačljiv program AKoff Music Composer, ki posneto WAVE datoteko pretvori v MIDI datoteko. Vsako posamezno glasbeno MIDI datoteko preprosto predstavi kot 2-dimenzionalno tabelo, v kateri ena dimenzija predstavlja višino, druga pa dolžino note. S takšno predstavitvijo glasbenega dela lahko težavo primerjave dveh glasbenih del reši z metodo ujemanja podvzorca v določenem vzorcu. To doseže najbolje z algoritmom, poimenovanim SMBGT[3], ki

[1] NLS ali "Note-based Linear Scaling" je metoda, ki linearno razteguje oz. krči notno predstavitev zamrmranega vhodnega posnetka za namen določanja faktorja, pri katerem se dva posnetka najbolje ujemata.
[2] NRA ali "Note-based Recursive Align" metoda temelji na metodi NLS, vendar s to razliko, da ne modificira celotnega posnetka skupaj, ampak posnetek rekurzivno razdeli na manjše dele, da se ta s ciljnim bolje ujema.
[3] SMBGT ali "Subsequence Matching with Bounded Gaps-Range-Tolerances" je metoda, ki se ukvarja z iskanjem podvzorca v bazi z velikim številom vzorcev.

temelji na ideji dinamičnega programiranja[4]. Uporabniku so na voljo tudi drugi algoritmi iskanja, med katerimi sta tako Algoritem razdalje kot tudi Algoritem dinamičnega ukrivljanja časa [7]. SMBGT lahko upošteva vrsto napak, kot so na primer šum v ozadju ali manjkajoče note v katerem koli od vzorcev. Omeji lahko tudi dolžino iskanega vzorca ter najmanjše število ujemajočih se elementov. Dopušča lahko veliko ali nič napak, saj je zasnovan tako, da je izjemno prilagodljiv oz. nastavljiv.

Nobeden od ponujenih sistemov QBH ne ponuja za nas optimalne rešitve prve faze. Za transkripcijo zajete zamrmrane melodije je bilo torej treba najti drugačen način. Na področju iskanja osnovne frekvence posnetka glasbe ali govora obstaja algoritem YIN [8] ali pa njegova izboljšana različica – verjetnostni YIN (angl. Probabilistic YIN ali pYIN) [9]. Algoritem pYIN je uporabljen v programu Tony [10], ki služi kot orodje za interaktivno transkripcijo enoglasnih avdio posnetkov.

Univerzalni sistem QBH, ki bi bil hkrati preprost za uporabo ter dovolj natančen, za zdaj še ne obstaja. Največ potenciala kaže sistem QBH Hum-a-song, saj vsebuje zelo učinkovit iskalni algoritem SMBGT. Slednji je bil zasnovan prav za uporabo v sistemih QBH. Število vhodnih parametrov mu omogoči izjemno prilagodljivost glede na vrsto uporabnika ter vrsto podatkovne baze, vendar pa ima njegova prilagodljivost tudi slabo lastnost. Med številnimi parametri se uporabnik hitro izgubi, pri čemer kompleksnost uporabniškega vmesnika prav nič ne pomaga, saj je ta zasnovan za naprednejše uporabnike.

To diplomsko delo se torej ukvarja najprej z iskanjem oz. zasnovo najoptimalnejšega sistema QBH za uporabo v spletni aplikaciji EtnoFletno, pozneje pa še z njegovo implementacijo. Največji potencial je pokazal sistem Hum-a-song z implementacijo iskalnega algoritma SMBGT, ki pa potrebuje dodatno optimizacijo za doseg želene učinkovitosti. V fazi transkripcije bo uporabljen algoritem pYIN, ki se je izkazal kot zelo natančen algoritem iskanja osnovne frekvence avdio posnetka.

[4] Dinamično programiranje je metoda, ki problem razdeli na manjše podprobleme, da zmanjša tako prostorsko kot tudi časovno zahtevnost algoritma.

Poglavje 2. Ozadje

2.1 Motivacija

Iskanje primernega sistema QBH se je zaradi specifičnosti področja izkazalo kot zelo zahtevno. Med obstoječimi je najprimernejši sistem Hum-a-song, saj je izjemno prilagodljiv in ima zato velik potencial. To diplomsko delo se osredotoča na implementacijo podobnega sistema. Cilj je izbor in optimizacija najučinkovitejšega algoritma za fazo transkripcije ter optimizacija parametrov najbolj učinkovitega algoritma za fazo iskanja najbolj podobnih vzorcev. Vse to z bazo, polno slovenskih ljudskih pesmi. Ciljni sistem mora biti za uporabnika preprost za uporabo ter kar se da učinkovit pri identifikaciji pravilne slovenske ljudske pesmi.

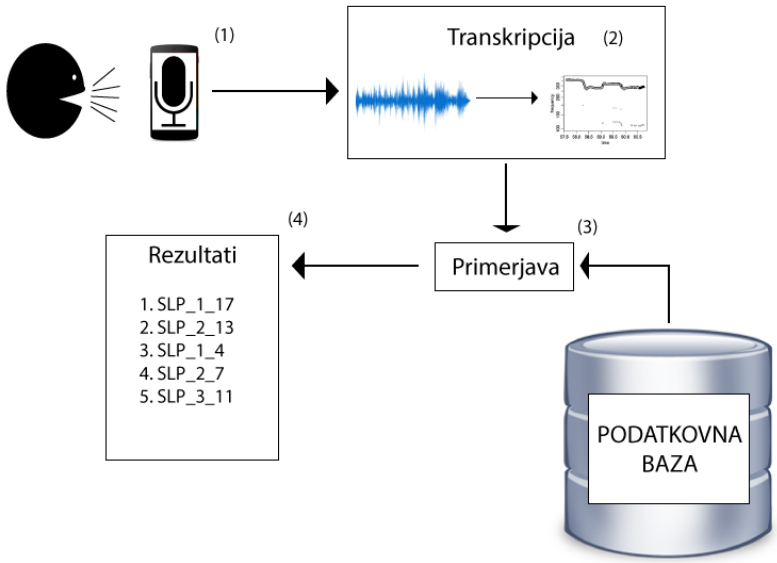
2.2 Pregled uporabljenih algoritmov

V diplomskem delu se uporabljeni algoritmi delijo na dve skupini. V prvi skupini so algoritmi, ki so uporabljeni v fazi transkripcije zamrmranega vzorca v zapis višine frekvence v odvisnosti od časa. To sta algoritma YIN ter njegova izboljšana različica pYIN, pri katerem začetnica predstavlja temelj za njegovo izboljšavo, upoštevanje verjetnosti (angl. Probabilistic YIN). Oba sta natančneje opisana v Poglavju 3.

Drugo skupino uporabljenih algoritmov predstavljajo algoritmi za iskanje podobnosti med glasbenimi vzorci. V Poglavju 3 so natančneje opisani algoritem Dinamičnega ukrivljanja časa (angl. Dynamic Time Warping ali DTW), algoritem Razdalje urejanja (angl. Edit distance), algoritem Spring, ki je izboljšana različica DTW-ja ter algoritma SMGT (ali angl. Subsequence Matching with Gaps-Range-Tolerances) in SMBGT (ali angl. Subsequence Matching with Bounded Gaps and Tolerances).

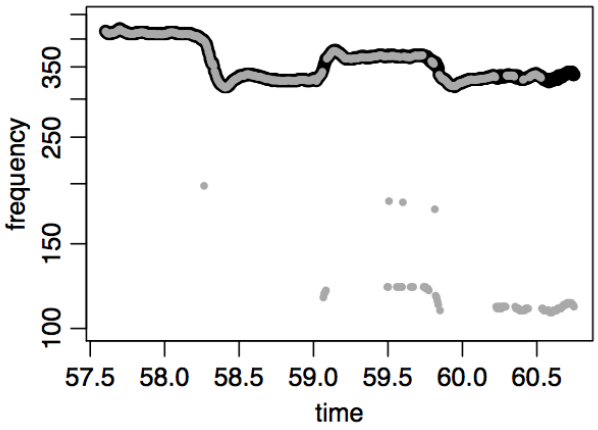
2.3 Struktura sistema QBH

Naloga sistema QBH je identifikacija vhodnemu zamrmranemu vzorcu najbolj podobnih glasbenih del v podatkovni bazi. Postopek identifikacije se začne s transkripcijo zamrmranega vzorca, od katere je odvisna uspešnost algoritmov za iskanje podobnosti. Sistem na koncu vrne omejeno število najbolj podobnih glasbenih del iz celotne podatkovne baze.



Slika 1: Pregled sistema QBH.

V diplomski nalogi je implementirani sistem QBH prikazan na Sliki 1. Zasnovan je tako, da v prvem koraku (1) posname uporabnikovo mrmranje oz. petje. Sistem potem izvede transkripcijo (2), v kateri iz signala v valovni obliki pretvori v zapis frekvenca v odvisnosti od časa. Primer končnega rezultata transkripcije je podan na Sliki 2. V tretjem koraku (3) sistem izvede primerjavo vhodnega signala z vsemi glasbenimi deli iz podatkovne baze. Sistem v četrtem koraku (4) vrne uporabniku nekaj glasbenih del, najbolj podobnih vhodnemu vzorcu.



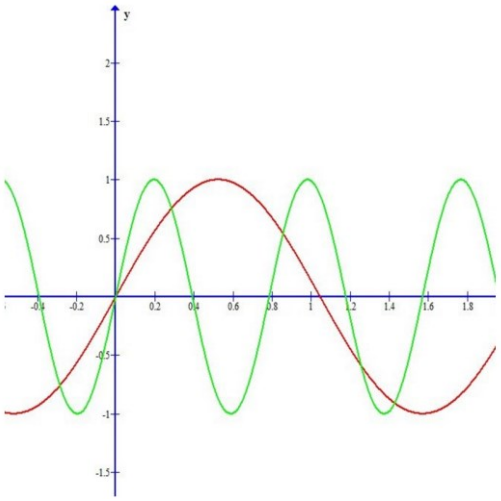
Slika 2: Ocena višine osnovne frekvence pri algoritmu pYIN (črna barva) ter YIN (siva barva). [9]

Poglavje 3. Uporabljeni algoritmi

3.1 Algoritmi za prepoznavanje osnovne frekvence

V tem poglavju bodo razloženi osnovni pojmi o glasbeni teoriji, ki so potrebni za nadaljnje razumevanje ter algoritma YIN in njegova izboljšana različica pYIN. Drugi korak diplomske naloge torej predstavljata ti dve metodi za prepoznavanje višine tona iz avdio signala. Vhodni zvočni posnetek je treba najprej s pomočjo teh dveh metod pretvoriti iz zaporedja bajtov v smiselne podatke, ki jih bodo lahko potem algoritmi za iskanje podobnosti uspešno procesirali. Iz vhodnih podatkov je treba izluščiti število vsebujočih not, vključno z njihovimi višinami, v hertzih, ter dolžino trajanja v milisekundah.

Višina tona je običajno izražena s frekvenco, ki predstavlja število nihajev v eni sekundi. Frekvenca ima mersko enoto hertz (Hz). Višina komornega tona - *a1*, ki se uporablja za uglaševanje glasbil v orkestru, niha s frekvenco 440 Hz. Večja, kot je frekvenca, višji je ton, kar dobro prikazuje spodnja slika.



Slika 3: Valovanje pri visokem tonu (zelena barva) in valovanje pri nižjem tonu (rdeča barva).

Algoritmi za prepoznavanje višine tona se uporabljajo za različne namene, kot npr. v fonetiki, kodiranju govora in pridobivanju informacij iz glasbe, zaradi česar se med seboj močno razlikujejo. Za zdaj še ne obstaja univerzalni algoritem, ki bi na vseh različnih področjih uspešno pripeljal do pravilnih rezultatov, obstaja pa zato več različnih algoritmov, ki jih v grobem razvrščamo glede na tri različne pristope [11].

Prvi je pristop s časovne domene, pri katerih se algoritmi ukvarjajo s problemom ocenjevanja osnovne frekvence s pomočjo oblike valovanja avdio signala, ki predstavlja spremembe zračnega pritiska

skozi čas. Teorija, ki se skriva za večino takšnih algoritmov, temelji na ugotavljanju, kako pogosto se oblika valovanja ponovi, s pomočjo katerega je potem ocenjena periodičnost posameznega signala. Če je signal periodičen, se lahko število ponovitev prešteje. Inverz števila ponovitev v sekundi nas pripelje do osnovne frekvence signala. Takšen pristop je izjemno preprost tako za razumevanje kot implementiranje.

Poleg že omenjenega ocenjevanja periodičnosti signala spada pod pristope s časovne domene tudi avtokorelacijska metoda. Korelacija med dvema oblikama valovanja avdio signala predstavlja njuno medsebojno podobnost. Avtokorelacijska metoda primerja signal sam s seboj v različnih časovnih intervalih, pri čemer je v primeru periodičnega signala periodična tudi sama avtokorelacijska funkcija. Prvi vrh v avtokorelaciji predstavlja periodo oblike valovanja. Slabost avtokorelacijske metode se pokaže pri harmonično kompleksnih oblikah valovanja, pri katerih se robustnost metode zmanjša, saj se poveča število dobljenih rezultatov. S tem se poveča računska kompleksnost metode, saj mora algoritem med vsemi temi vrhovi razlikovati, da lahko iz prvega največjega dobimo osnovno frekvenco signala. Izboljšava avtokorelacijske metode je algoritem YIN, ki tudi spada med pristope s časovne domene.

Algoritmi za zaznavanje osnovne frekvence v signalu so lahko razviti tudi s pomočjo metod s frekvenčne domene [11]. V frekvenčni domeni je veliko informacij, ki so povezane z osnovno frekvenco signala. Signali z določeno višino so po navadi sestavljeni iz zaporedja harmoničnih tonov, ki se jih lahko identificira in uporabi pri izračunu osnovne frekvence. Od pristopov s časovne domene so bolj natančni, vendar časovno zahtevnejši.

Tretji pristop je časovno spektralni, iz katerega je bil razvit najbolj znan takšen algoritem YAAPT (ali angl. Yet Another Algorithm for Pitch Tracking) [12]. Dobra lastnost algoritma je njegova robustnost pri procesiranju tako visoko kot tudi manj kakovostnih signalov (npr. telefonski govor). Algoritem najprej poišče vse kandidate osnovnih frekvenc s pomočjo metode NCCF (ali angl. Normalized Cross Correlation) [13], nato z informacijami, pridobljenimi iz spektrograma, izlušči tiste, ki so najbolj verjetni. S pomočjo metode dinamičnega programiranja, ki vsak problem razdeli na manjše podprobleme, na koncu algoritem skonstruira zaporedje frekvenc tonov, ki predstavljajo vhodni zvočni signal.

3.1.1 Prepoznavanje osnovne frekvence periodičnega signala (F_0)

Višina zvoka je po navadi odvisna od njegove osnovne frekvence. Običajno tudi velja, da se periodičnim zvokom da določiti tudi njihovo višino, vendar prav tako obstajajo izjeme. Pojem višina zvoka (angl. sound pitch) je pogosto uporabljen namesto osnovne frekvence, zaradi česar metode prepoznavanja osnovne frekvence pogosto imenujemo algoritmi za prepoznavanje višine zvoka (angl. pitch detection algorithms ali PDA [14]). Moderni modeli zaznavanja višine tona predvidevajo, da ta izvira iz periodičnosti nevronskih vzorcev v časovni domeni [15] [16] [17] [18] ali pa iz vzorca harmoničnih tonov, ki jih v našem telesu zazna notranje uho v frekvenčni domeni [19] [20] [21]. S pomočjo obeh metod dobimo enake rezultate, in sicer osnovno frekvenco ali njen inverz oz. periodo [22].

Zaradi premikanja vokalnega trakta, ki filtrira valovno dolžino z glasilkami proizvedenega zvoka, lahko periodične vibracije glasilk proizvedejo govor, ki ni popolnoma periodičen. Že same vibracije glasilk lahko pokažejo znake aperiodičnosti, kot so spremembe amplitude, frekvence ali valovne dolžine. Vse skupaj negativno vpliva na uporabnost rezultatov algoritmov in metod.

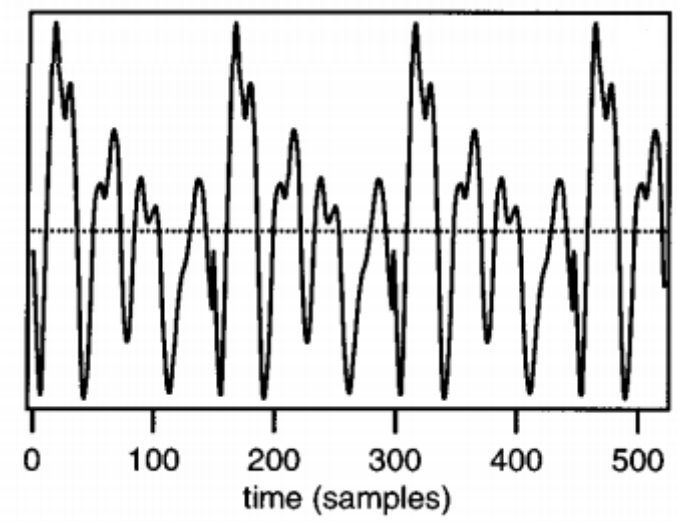
Metode, ki nam omogočajo učinkovito ugotavljanje osnovne frekvence zvočnega posnetka, se lahko uporabljajo za različne namene:

- odkrivanje vzorcev ritma in zvokov, uporabljenih v poeziji,
- odkrivanje govora,
- avtomatizirana transkripcija glasbe,
- zbiranje ostalih multimedijskih metapodatkov.

3.1.2 Algoritem YIN

Algoritem YIN sta razvila francoski znanstvenik Alain de Chegeigne in profesor z japonske univerze Wakayama Hideki Kawahara. Njegov glavni namen je ugotavljanje osnovne frekvence avdio posnetka oz. zvočnega signala. Uporablja se tako v glasbi kot pri govoru, saj iskana frekvenca ni omejena navzgor. Temelji na že znani avtokorelacijski metodi, ki pa jo dopolnjuje s številnimi spremembami za preprečitev napak [8]. Po številnih preizkusih z bazo, ki je vsebovala posnetke govora, je imel algoritem trikrat manj odstopanja in napak v primerjavi z drugimi metodami na tem področju. Implementacija je relativno preprosta in je izvedljiva z zelo majhno zakasnitvijo, prav tako pa vsebuje tudi število nastavljivih parametrov. Temelji na modelu signala, ki se ga da modificirati glede na aperiodičnost signala v dani implementaciji.

Metoda je sestavljena iz šestih glavnih korakov. Prvi korak sestoji iz avtokorelacijske metode, ki primerja signal same s seboj v različnih časovnih intervalih. Rezultat funkcije je signal, katerega vrhovi so na večkratnikih njegove periode. Z analizo vseh rezultatov metoda izbere najvišji vrh. V nekem smislu naredi podobno kot metoda AMDF [5]. Avtokorelacijska funkcija je Fourierova transformacija moči spektra, ki si jo lahko predstavljamo kot merjenje povprečnega razmika med harmoniki v danem spektru. Metoda v veliko primerih naredi napake, zato so naslednji koraki namenjeni zmanjševanju teh napak.



Slika 4: Primer oblike valovanja signala v odvisnosti od časa. [8]

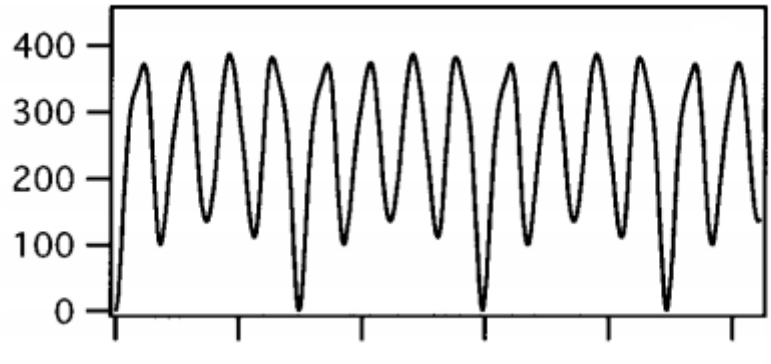
V drugem koraku algoritma se izboljša napake, do katerih privedejo signali, ki imajo velike nihaje v amplitudi. Vrhovi amplitud z avtokorelacijsko funkcijo bi morali ostati konstantni, vendar se s povečanjem amplitude vhodnega signala s časom dvignejo, kar privede do prenizkih izračunov osnovne frekvence. Ravno obratno se zgodi pri znižanju amplitude s časom. Uporaba funkcije razlike

[5] Metoda AMDF (angl. »Average Magnitude Difference Function«) izvede primerjavo med signalom ter njegovo zamaknjeno različico s pomočjo razlik namesto produktov, kot to počne avtokorelacijska metoda [AMMDF]. (Ross et al., 1974; Ney, 1982)

je pri odpravljanju teh napak zelo učinkovita, saj je neodvisna od sprememb v amplitudi. Iz nje sledita tudi naslednja koraka, ki se ukvarjata s prenizkimi in previsokimi izračuni osnovne frekvence.

$$d_t(\tau) = \sum^{W} (x_j - x_{j+\tau})^2 \tag{1}$$

V zgornji funkciji razlike (1) W predstavlja velikost okna, črka τ pa zamik (angl. lag).

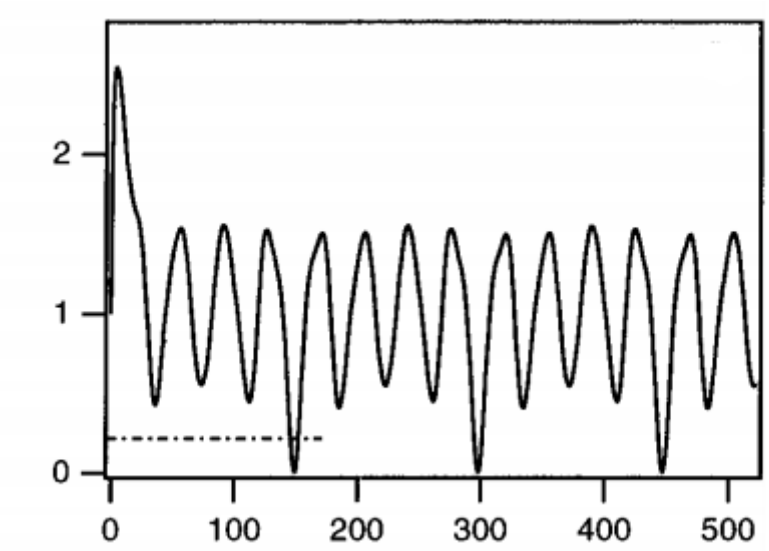


Slika 5: Rezultat funkcije razlike za signal iz prejšnje slike. [8]

Z izračunom kumulativne povprečne normalizirane funkcije razlike se vrednosti, pridobljene v prejšnjem koraku, deli s povprečjem vsote njenih vrednosti v kratkem časovnem intervalu. V enačbi (2) črka τ predstavlja zamik.

$$d'_t(\tau) = \begin{cases} 1, & pri\ \tau = 0, \\ \dfrac{d_t(\tau)}{(1/\tau)\sum_{j=1}^{\tau} d_t(j)}, & drugače \end{cases} \tag{2}$$

Tako se algoritem znebi previsokih izračunov osnovne frekvence in hkrati normalizira vrednosti funkcije za uporabo v naslednjem koraku.



Slika 6: Rezultat izračuna kumulativne povprečne normalizirane funkcije razlike. Iz slike je dobro razvidno, da se namesto pri nič začne pri ena in ostane visoko, dokler ne doseže prvega minimum pri periodi signala. [8]

Pri vrednosti zamika nič, je funkcija razlike, prikazana na zgornji sliki, tudi enaka nič. Pri vsaki periodi signala pa je le ta pogosto večja od nič, kar nakazuje na nepopolno periodičnost vhodnega signala. Če spodnji limit ni nastavljen, je algoritem prisiljen izbrati periodo na mestu, pri katerem je rezultat funkcije razlike prvič enak nič. Tako se pogosto zgodi, da je izbrana perioda napačna, kar privede do prenizkega izračuna osnovne frekvence signala. V četrtem koraku tako algoritem uvede spremenljivko absolutnega praga. Ta skrbi, da funkcija razlike nikoli ne vrne vrednosti, ki je od njega nižja, kar zmanjša število napačnih ocen periode. Z vrednostjo spremenljivke 0.1 se zmanjša število napačnih izračunov že za dobro polovico.

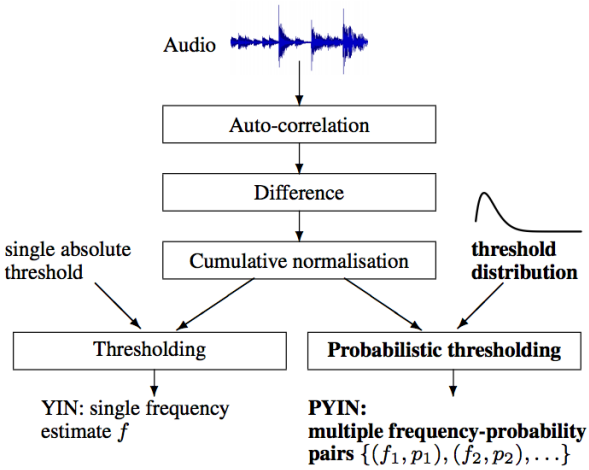
Za vsak pridobljeni lokalni minimum iz prejšnjega koraka algoritem v petem koraku najde najbolj ustrezno parabolo tako, da se na njej nahaja največje možno število lokalnih minimumov. S pomočjo parabolične interpolacije se nato oceni nov minimum, ki ga abscisa pozneje služi za oceno periode. Takšna metoda je včasih lahko rahlo pristranska, zato se za končno oceno periode vzame vrednost istoležnega minimuma, pridobljenega iz prvotne diferenčne funkcije v drugem koraku.

Sledi le še zadnji korak, v katerem ob vsaki izračunani oceni iz petega koraka algoritem še preišče njeno okolico, ali se morda v njej nahaja potencialno boljši kandidat za lokalni minimum. Korak spominja na glajenje z mediano ali celo na tehnike dinamičnega programiranja, vendar se razlikuje v tem, da vzame v račun relativno kratek interval ter hkrati daje poudarek pri odločitvi bolj na kakovost rezultata kot njegovo kontinuiteto.

3.1.3 Algoritem pYIN

Izboljšavo algoritmu YIN sta predlagala Matthias Mauch in Simon Dixon z Univerze Queen Mary v Londonu. Pomanjkljivost YIN algoritma je v tem, da četrti korak, v katerem se postavi absolutni prag, poda le en rezultat glede na trenutni okvir. To pomeni, da so vsi ostali potencialni kandidati zavrženi in se jih v poznejšem procesiranju algoritma ne mora več upoštevati. Izkazalo se je, da bi z upoštevanjem vseh kandidatov v zadnjih dveh korakih algoritma lahko še izboljšali natančnost dobljenih rezultatov. Izboljšan algoritem pYIN spremeni korak originalnega algoritma tako, da upošteva vse potencialne kandidate, ki jim doda še njihovo verjetnost. Od tu izhaja tudi ime - verjetnostni (angl. probabilistic) YIN. Z dodajanjem verjetnosti se tako pred glajenjem rezultatov izogne izgubljanju preveč potencialno pomembnih informacij. Z novo pridobljenimi rezultati pYIN uporabi tehniko Skritih Markovih Modelov [6] v kombinaciji z Viterbi algoritmom [7] za izračun končnega rezultata - najverjetnejše osnovne frekvence [9].

V prvem delu se algoritma razlikujeta pri določitvi praga. YIN predpostavi, da je prag le en in tako nikoli ne izbere nižje vrednosti. Algoritem pYIN v tem koraku upošteva vse potencialne kandidate, ki jim doda še njihove verjetnosti, in tako prag interpretira kot spremenljivko s svojo porazdelitvijo. S tem vedno pokrije vse možnosti dobljenih osnovnih frekvenc originalnega algoritma, ki jim doda še tiste z uvedbo absolutnega praga.



Slika 7: Primerjava prvih korakov originalnega algoritma YIN z njegovo izboljšano različico pYIN, ki je označena z odebeljenim besedilom. [9]

Čeprav pYIN upošteva več kandidatov, se njegova časovna zahtevnost v primerjavi z originalnim algoritmom poveča zelo malo. Če YIN izbere napačnega kandidata v četrtem koraku pri določanju praga, bomo vedno dobili nepravilni končni rezultat. Dokazano je bilo, da je v veliko primerih

[6] Tehnika Skritih Markovih Modelov (angl. Hidden Markov Model ali HMM) se uporablja pri prepoznavanju vzorcev v glasbi, pisanju, gibih itd. V teh primerih so rezultat ter verjetnosti vmesnih korakov že poznani, ugotavlja pa se točna pot do končnega rezultata.

[7] Za dekodiranje SMM se uporablja Viterbi algoritem, ki nam poda najverjetnejše zaporedje skritih stanj, poimenovano Viterbi pot. Ta nas vodi do poznanega končnega rezultata.

obstajala neznana vrednost praga, ki bi tudi pri originalnem algoritmu vodila do pravilnega rezultata, vendar jo je le ta zavrgel. To je bila glavna pobuda za izboljšavo originalnega algoritma.

Zadnja koraka sta zasnovana za izbiro največ enega od vseh možnih kandidatov za osnovno frekvenco v določenem časovnem okvirju. Pri algoritmu pYIN se to izvede s pomočjo tehnike Skritih Modelov Markova (pozneje SMM). Vse možne osnovne frekvence se razdeli na 480 košev, ki obsegajo štiri oktave, od tona A1 (55 Hz) do tona A5 (880 Hz), s korakom po 0.1 poltona. Omenjene koše je možno modelirati kot stanja v SMM. Model potem uporabi verjetnosti kandidatov za osnovno frekvenco kot opazovano vrednost. Verjetnost vsakega kandidata je dodeljena košu, ki je najbližji oceni za njegovo osnovno frekvenco. Da je model še bolj stvaren, se dodeli vsaki možni osnovni frekvenci tudi njeno stanje, ki je lahko zveneče ali nezveneče. Prehodne verjetnosti v modelu imajo dva glavna namena: dajanje prednost naravnejšim (nepretrganim) zaporedjem not ter dajanje prednosti manjšemu številu sprememb zvena. Pri izračunu prehodnih verjetnosti med dvema stanjema, definiranima z zvenom ter višino osnovne frekvence, se predpostavi, da sta zven in višina osnovne frekvence neodvisna. Izračun tako predstavlja kar rezultat med individualnima verjetnostma. Začetne verjetnosti so enakomerno porazdeljene po stanju zvena. Model je potem dekodiran z učinkovito različico Viterbi algoritma.

Algoritem pYIN je dokazano boljši od svojega predhodnika. Prvotni algoritem YIN ima mediano uspešnosti ocen glavne frekvence 0.951, njegova izboljšana različica pa kar 0.98. Izboljšava se pozna tudi pri številu napak pri oceni oktave ter zvena. Ocene oktave so tako nepravilne v povprečno 1,03 % primerov, zven pa je pravilno ocenjen v kar 93,87 % primerov.

3.2 Algoritmi za iskanje podobnosti

Omenjenih bo pet algoritmov, ki se jih uporablja pri iskanju podobnosti med dvema poljubnima sekvencama. V diplomski nalogi se uporabljajo pri primerjavi vhodnega zamrmranega signala z vsakim posnetkom, ki se nahaja v podatkovni bazi. Algoritem Dinamičnega ukrivljanja časa (angl. Dynamic Time Warping ali pozneje DTW) ter algoritem Razdalje urejanja sta med njimi najbolj poznana in se ju najpogosteje uporablja pri iskanju podobnosti med dvema besedama. Oba sta ustrezno predelana, da sprejmeta s parametri vektor števk, ki predstavlja vhodni signal ter dolg vektor vseh posnetkov iz podatkovne baze. Algoritem Spring je izboljšana različica algoritma DTW, saj se slednji ni dobro obnesel na področjih, na katerih se podatkovna baza s časom povečuje. Izboljšava ga naredi izjemno učinkovitega na področju omrežne analize. Ker pa je v našem primeru baza statična, se ne odreže nič bolje od algoritma DTW. Zadnji algoritem SMGT ter njegova izboljšana različica sta bila razvita prav z namenom uporabe v sistemih, ki se ukvarjajo z iskanjem v podatkovni bazi s pomočjo zamrmranega avdio vzorca. Zaradi prilagodljivosti, ki jo omogoča številčnost vhodnih parametrov, sta se za namen uporabe v diplomski nalogi izkazala kot najuspešnejša.

3.2.1 Dynamic time warping

DTW je tehnika iskanja najbolj optimalne poravnave dveh časovno neodvisnih sekvenc z določenimi omejitvami. Sekvenci sta nelinearno zviti, da se prilegata druga drugi. Sprva je bil DTW uporabljen za primerjavo govornih vzorcev v samodejni zaznavi govora, vendar se je sčasoma začel uporabljati tudi na drugih področjih, kot so podatkovno rudarjenje in iskanje informacij [7]. Tehnika je bila uspešno uporabljena za samodejno spopadanje s časovnimi deformacijami in različnimi hitrostmi določenih časovno odvisnih podatkov.

V splošnem meritev podobnosti predstavlja funkcija $Podobnost\ (X, Y)$, kjer sta X in Y vzorca istega tipa podatka. Vrednost funkcije $Podobnost\ (X, Y)$ leži v intervalu $[0,1)$. Večja kot je vrednost, bolj sta si vzorca podobna, pri čemer $Podobnost\ (X, Y) = 0$ pomeni, da si vzorca med seboj nista nič podobna. Za preprost časovni interval lahko meritev podobnosti predstavimo kot korelacijo koeficientov ali njuno medsebojno kosinusno razdaljo. Ker je za kompleksen nabor podatkov težko natančno predstaviti podobnost z zgornjo funkcijo, je podobnost po navadi predstavljena kot medsebojna razdalja dveh vzorcev. Obstaja več različnih metod predstavitve razdalje podobnosti, med katerimi je največkrat uporabljena razdalja Minkowskega, ki je definirana kot:

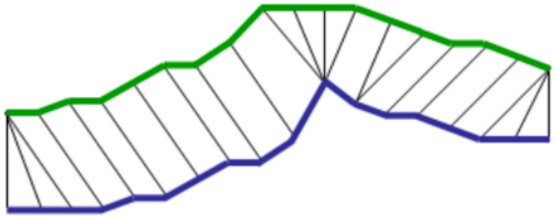
$$d(X, Y) = (\sum^{n} |x_i - y_i|^P)^{1/P} \tag{3}$$

Ko je $P = 2$, je medsebojna razdalja dveh vzorcev poimenovana evklidska razdalja. Iz zgoraj navedene enačbe vidimo, da je razdalja nič, ko sta vzorca popolnoma enaka. Z naraščajočo razdaljo narašča tudi medsebojna razdalja vzorcev. Pri računanju razdalje Minkowskega, je treba zagotoviti

enako dolžino obeh vzorcev in enako utež razlike med vsakim parom. Zaradi omenjenih pogojev razdalja Minkowskega ne more biti učinkovita pri računanju medsebojne razdalje kompleksnejših vzorcev s premikajočo in raztezajočo se amplitudo.

Predpostavimo, da imamo dva vzorca A in B, ki sta predstavljena vsak s svojim vektorjem. Njuni dolžini sta n in m.

$$A = a_1, a_2, \dots, a_i, \dots, a_n$$
$$B = b_1, b_2, \dots, b_j, \dots, b_m$$

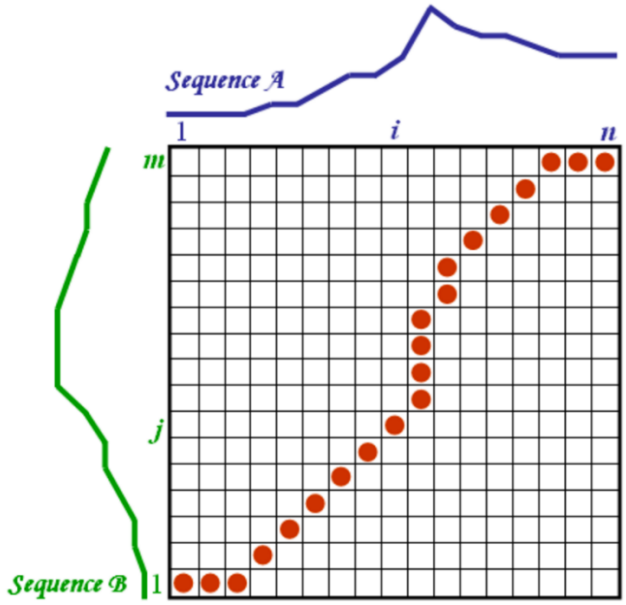


Slika 8: Predstavitev dveh vzorcev (zelen in moder), pri katerih je za namen optimalnega prileganja medsebojnih točk algoritem DTW skrivil časovno os. [23]

Vsak vektor ima d dimenzij in je zato lahko predstavljen v d dimenzionalnem prostoru. Pri prepoznavi pisave lahko na primer koordinate svinčnika (x, y) neposredno predstavimo kot dvodimenzionalni vektor. Na takšen način bi s pisanjem kreirali sekvenco teh dvodimenzionalnih vektorjev. V praksi bi seveda uporabili več uporabnih lastnosti kot le koordinate in bi kreirali vektor z več kot dvema dimenzijama. S to metodo popolnoma izničimo pomen časovne komponente, kar pomeni, da dolžini vektorjev ne vplivata na rezultat. Metoda DTW krivi časovno os iterativno, dokler ne doseže optimalne podobnosti dveh vzorcev.

Za izračun najoptimalnejše poti lahko naredimo matriko razdalje $n * m$. V matriki vsaka celica (i, j) predstavlja razdaljo med i-tem elementom vzorca A in j-tem elementom vzorca B Za metriko je uporabljena evklidska razdalja.

Iskanje najboljše poravnave dveh sekvenc si lahko predstavljamo kot iskanje najkrajše poti z začetne točke levo spodaj do točke, ki je na spodnjem grafu v desnem zgornjem kotu. Dolžina poti je preprosto kar seštevek celic, ki so bile obiskane na poti od začetne do končne točke. Dlje, kot gre optimalna pot od diagonale, bolj je treba vzorca skriviti, da se ujemata.



Slika 9: Iskanje najkrajše poti z začetne točke levo spodaj do končne desno zgoraj. [23]

3.2.1.1 Dinamično programiranje

S pomočjo dinamičnega programiranja lahko izkoristimo omejitve in najdemo najboljšo pot na rekurziven način namesto z grobo silo. Prej je bila celica $(i,\ j)$ matrike razdalj definirana kot razdalja med i elementom vzorca A ter j elementom vzorca B, dinamično programiranje pa to redefinira, in sicer na način, da celica $(i,\ j)$ predstavlja dolžino najkrajše poti do te celice. Zdaj je celica definirana kot sledeča enačba (4).

$$celica(i,j) = local\_distance(i,j) + MIN(celica(i-1,j), celica(i-1,j-1), celica(i,j-1)) \quad (4)$$

Z rekurzivne perspektive je najkrajša pot do celice (i,j) definirana kot najkrajša pot do sosednjih celic. Na tem mestu lahko uporabimo veliko lokalnih omejitev, zato se implementacije DTW-ja tukaj razlikujejo. Pri končni celici se algoritem ustavi in za ugotovitev najoptimalnejše poti uporabi algoritem vračanja (angl. backtracking). Če je naša naloga le primerjava dveh vzorcev, zadnja celica matrike predstavlja tudi dolžino najoptimalnejše poti. Algoritem DTW je simetričen, zato velja $DTW(a,b) = DTW(b,a)$, ne upošteva pa trikotne neenakosti, kar v praksi ni težava. Zelo je podoben Levenshteinovem algoritmu, ki se uporablja za primerjavo besed.

3.2.2 Edit Distance - razdalja urejanja

V računalništvu nam razdalja urejanja pove, kako različni sta si med seboj dve besedi. Razdalja predstavlja minimalni seštevek operacij, ki jih potrebujemo, da spremenimo eno besedo v drugo [24]. Malo prirejen algoritem se lahko uporabi pri primerjavi nebesednih struktur, kot je na primer zaporedje števil oz. številski vektor. Obstaja več različnih implementacij metode iskanja razdalje urejanja, pri čemer je najbolj znana Levenshteinova, ki je dobila ime po ruskem znanstveniku s področja računalništva, Vladimirju Levenshteinu. Njegova implementacija dovoljuje operacije dodajanja in odvzemanja znakov ter zamenjave enega znaka za drugega. Zaradi popularnosti Levenshteinove implementacije lahko pojma med seboj enačimo.

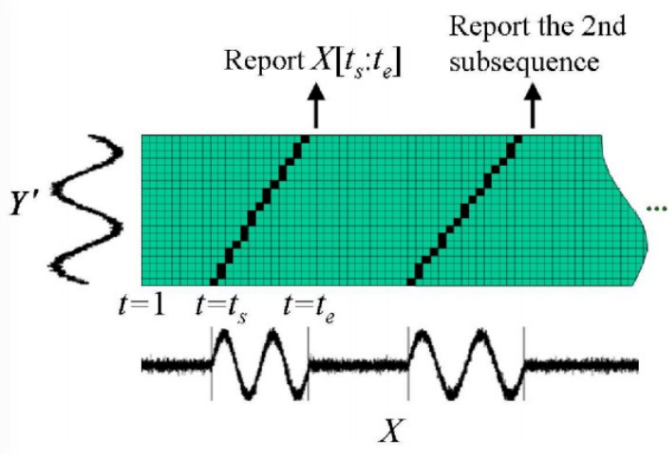
Psevdokoda spominsko učinkovitejše implementacije algoritma za računanje razdalje urejanja, ki ima časovno zahtevnost $O(2 * min(Dolžina1, Dolžina2))$ $namesto$ $O(Dolžina1 * Dolžina2)$ [4]:

1. Nastavimo parametre
    a. s je prva beseda
    b. t je druga beseda
    c. n je dolžina s-ja
    d. m je dolžina t-ja
        i. če je $n = 0$; vrnemo m in končamo
        ii. če je $m = 0$; vrnemo n in končamo
    e. zgradimo dva vektorja $v_0[m + 1]$ ter $v_1[m + 1]$, ki vsebujeta 0 ... m elementov
2. Vzpostavimo $v_0 = 0 \ldots m$
3. Pregledamo vsak znak s-ja ($i = 1; i <= n$)
4. Pregledamo vsak znak t-ja ($j = 1; j <= m$)
5. Če je $s[i]$ enak $t[j]$, je razdalja 0
6. Če s[i] ni enak $t[j]$, razdalji dodamo 1
7. Nastavimo celico $v_1[j]$ enako minimumu od:
    a) celici nad njo plus 1: $v_1[j - 1] + 1$
    b) celici levo od nje plus 1: $v_0[j] + 1$
    c) celici diagonalno levo zgoraj od nje plus razdaljo: $v_0[j - 1] + razdalja$
8. Ko so koraki 3, 4, 5, 6 končani, se končna razdalja nahaja v celici $v_1[m]$

3.2.3 Spring

Algoritem Spring je predlagala skupina znanstvenikov kot izboljšavo algoritma dinamičnega ukrivljanja časa. Algoritem DTW se uporablja za merjenje razdalje med vzorci, saj je njegova ključna lastnost ta, da izniči pomembnost časovne komponente z ukrivljanjem časa [25]. Ima pa pomanjkljivost, ki je najvidnejša na področju omrežne analize. Ker najbolje deluje pri končnem številu shranjenih vzorcev v podatkovni bazi, je neprimeren za uporabo v okoljih, kjer se ta baza s časom povečuje. Pri omrežni analizi je tok podatkov množičen, obenem pa tudi neprekinjen, kar onemogoči shranjevanje vseh prejetih podatkov v optimalnem času, na kar pa se DTW zanaša. Za takšen namen je bil predlagan algoritem Spring. Z eksperimenti so dokazali, da predlagan algoritem uspešno pride do pravilnih rezultatov, pri čemer je časovna zahtevnost znatno manjša od algoritma DTW.

Izboljšava temelji na dveh idejah. Originalen algoritem zgenerira v vsaki časovni enoti novo matriko razdalj s časovno zahtevnostjo $O(n)$. Prva ideja izboljša časovno zahtevnost, tako da se hrani samo ena matrika razdalj podvzorcev prvega vzorca, drugemu vzorcu dodamo na začetek element zvezdni element, ki predstavlja interval $(-\infty : +\infty)$ in vedno vrne razdaljo 0. Drugi vzorec s tem dodanim elementom uporabimo za izračun razdalje med podvzorci drugega vzorca in podvzorci prvega vzorca. To nam omogoči uporabo le ene matrike razdalj pri iskanju ustreznega podvzorca prvega vzorca. Zvezdno oblazinjenje (angl. star padding) dramatično izboljša časovno in prostorsko zahtevnost iz $O(nm)$ na $O(m)$, saj potrebujemo posodobiti v časovni enoti le m-številk. Končni rezultat koraka nam poda konec ujemajočega podvzorca ter njegovo razdaljo od vzorca v poizvedbi. V večini primerov uporabe takšnih algoritmov je potreben tudi časovni žig začetka ujemajočega se podvzorca glede na celoten vzorec, na čemer temelji druga ideja algoritma Spring. Matrika razdalj podvzorca (angl. Subsequence time warping matrix ali STWM) vsebuje dve vrednosti vsakega podvzorca. Njegovo razdaljo ter časovni žig začetka. Matrika se gradi med procesiranjem algoritma, najbolj uporabna pa je za hitro identifikacijo podvzorca, ki ustreza našim trenutnim argumentom. Izboljšava algoritma Spring torej temelji na kombinaciji obeh idej, ki omogočata učinkovit izbris vseh podatkov o nepotrebnih podvzorcih z uporabo le ene matrike.



Slika 10: Algoritem Spring je zasnovan tako, da učinkovito zazna vzorce s podatkovnega toka, ki so si med seboj najbolj podobni. [25]

3.2.4 Algoritma SMGT in SMBGT

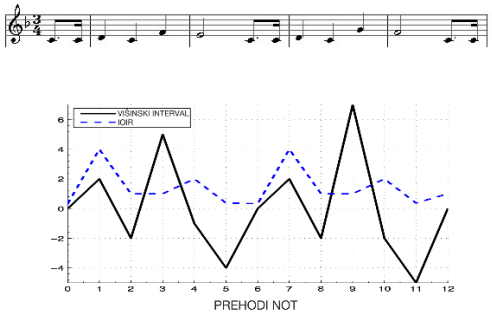
Omenjeni so bili že trije algoritmi, ki implementirajo rešitev za problem iskanja podvzorca v bazi z velikim številom vzorcev, ki se najbolje ujema z določeno poizvedbo. Od tega dva temeljita na ideji dinamičnega programiranja - DTW ter Spring - ki se izkaže kot dober način obdelovanja velike količine podatkov na učinkovit način. Z razdorom problema na manjše podprobleme ne le zmanjšamo časovno, ampak tudi prostorsko zahtevnost. Algoritmi se dobro obnesejo pri kategoričnih zaporedjih, multimedijskih podatkih, časovnih zaporedjih in tako dalje. Natančneje je tukaj govora o analizi glasbenih vzorcev, kjer pa algoritmi, ki niso zasnovani prav v ta namen, ne delujejo, ali pa je njihova uspešnost zelo majhna. SMBGT ter SMGT sta tudi algoritma, ki uporabljata metode dinamičnega programiranja. Njuna največja prednost pred drugimi je dejstvo, da sta zasnovana za rešitev zgoraj omenjenega problema prav na področju glasbe.

Alexios Kotsifakos, Panagiotis Papapetrou, Jaakko Hollmen in Dimitrios Gunopulos so Grški znanstveniki z Univerze v Atenah, ki so razvili algoritma SMBGT ter SMGT za namen uporabe v sistemih QBH. Glavna naloga takšnih sistemov je, da pri podani zamrmrani poizvedbi preiščejo celotno podatkovno bazo z namenom najdbe K - v naprej določenega števila želenih rezultatov - najbolj podobnih pesmi. Naloga je neposredno povezana z iskanjem podvzorca v bazi z velikimi številom vzorcev, saj je zamrmrana poizvedba tipično majhen del iskane melodije. Ker so se problema lotili z vidika glasbe, sta algoritma zasnovana s poudarkom na iskanju s podvzorcem, ki je v časovnem zaporedju [26].

Vsako glasbeno delo je zaporedje not, karakteriziranih z *glasbenim ključem* in *tempom*. Glasbeni ključ opredeljuje standardni vzorec dovoljenih intervalov, s katerimi mora biti v skladu tudi določeno zaporedje not. Tempo regulira hitrost glasbenega dela. Vsaka nota je sestavljena iz dveh delov: višine note (»pitch«) ter njene dolžine. Interval višine note je razdalja med dvema višinama različnih not. Najmanjši interval je poimenovan polton, skupek dveh poltonov je ton, interval dvanajstih poltonov pa se imenuje oktava.

Oktava predstavlja nihajno razmerje 2 : 1. Interval označuje višinsko razdaljo med dvema tonoma. Tudi oktava je interval. Druge intervale dobimo iz celoštevilskih nihajnih razmerij med dvema tonoma. Razmerju 2 : 1 (oktava) sledi 3 : 2, to je kvinta (peti ton tonske lestvice), nato 4 : 3, kvarta (četrti ton) in tako naprej. Celoštevilsko nihajno razmerje bistveno lažje zaznamo z ušesom kakor z razumom. Toni, do katerih pridemo po tej poti, sestavljajo tonsko lestvico. To je sosledje tonov med začetnim in končnim tonom oktave, ki ga tudi občutimo kot naravno zaporedje tonov. Ta naravnost je razvidna iz fizikalnih številskih razmerij med frekvencami.

Sprememba glasbenega ključa, pod katerim je napisana melodija, v drug ključ, se imenuje transpozicija. Vsako glasbeno delo je karakterizirano kot monofonično ali polifonično. Polifonično delo sestoji iz več hkrati potekajočih melodij, medtem ko je pri monofoničnem le ena melodija. Ker je mrmranje monofonična melodija, se pri QBH upošteva le monofonična glasbena dela.



Slika 11: Primer izseka glasbe in njegove dvodimenzionalne časovne predstavitve.

Višina ter dolžina tona sta ključna razlikovalna faktorja za vsako glasbeno delo in sta zato oba uporabljena v vsaki učinkoviti predstavitvi glasbe. Če bi imeli več pesmi, ki imajo zelo podobne višine tonov, bi nam algoritem brez upoštevanja dolžine tonov vračal napačne rezultate. Vsaka melodija, ki ju algoritma upoštevata, mora biti zato definirana kot dvodimenzionalno časovno zaporedje not poljubnih dolžin, pri čemer je ena dimenzija višina, druga pa dolžina tona.

Za zagotavljanje robustnega in smiselnega ujemanja podvzorcev v okolju s potencialno veliko šuma je treba upoštevati nekaj parametrov algoritma. Pri mrmranju melodije hitro nastanejo manjše ali celo večje napake, bodisi pri hitrosti bodisi pri višini, zato algoritma omogočata, da v neki meri tolerirata te napake. Poleg tega podpirata tudi preskakovanje elementov vzorca tako v poizvedbi kot v ciljnem vzorcu. Število preskokov je omejeno s številom *r,* kar prepreči algoritmu proizvodnjo zelo dolgih ujemajočih se podvzorcev, ki imajo velike vrzeli med ujemajočimi elementi. Algoritmu je treba nastaviti tudi najmanjše število ujemajočih se elementov, da bo ta našel podvzorec z največjim možnim ujemanjem. Z uporabo teh parametrov in poznavanjem pevskih sposobnosti osebe, ki je zamrmrala poizvedbo, lahko prilagodimo toleranco algoritma do napak v poizvedbi. Še več, algoritem SMBGT omogoča tudi omejitev števila dovoljenih zaporednih preskokov tako v poizvedbenem kot tudi ciljnem vzorcu. Poimenovana sta *alfa* in *beta* in predstavljata razliko med SMGT in SMBGT algoritmoma. Slednja parametra nadzirata tudi razširjanje ujemajočih podvzorcev med izračuni dela algoritma, ki je implementiran z metodo DP.

Predstavitvene sheme, razvite za iskanje podvzorcev v glasbenem delu, po navadi predstavljajo noto samo z njeno višino. Veliko več informacije o noti pa lahko pridobimo in pozneje uporabimo, če v predstavitev vključimo tudi njeno dolžino. V vsaki predstavitveni shemi lahko višino tona predstavimo na dva načina:

1. absolutna višina
    - uporabi se frekvenca note
    - pri MIDI datoteki je to številka med 1 in 127, kjer 0 predstavlja pavzo
2. višinski interval
    - razlika v frekvenci med dvema sosednjima notama

Dolžina note se lahko predstavi na tri različne načine:

1. Inter-Onset-Interval (IOI)
    - razlika v času med začetkoma dveh sosednjih not
2. Inter-Onset-Ratio (IOIR)
    - razmerje razlike v času med začetkoma dveh sosednjih not, kjer je IOIR zadnje note enak 1
3. Log IOI Ratio
    - logaritem IOIR

Za dvodimenzionalno predstavitev časovnega zaporedja je pri uporabi algoritmov SMBGT in SMGT mogoča katera koli kombinacija zgornjih predstavitev, vendar je predlagana uporaba <višinski interval, IOIR> in <višinski interval, LogIOIR>. Z uporabo takšnih kombinacij imamo opravka le s spremembo višin not, s čimer prihranimo pri časovni zahtevnosti izračunov, saj nam ni treba analizirati vseh transpozicij note. Prav tako so višinski intervali kvantizirani v interval [-11, 11] z ostankom pri deljenju z 12 (mod 12). Takšna kvantizacija ustreza dvema oktavama, kar je klasičen razpon višin tonov pri človeškem petju.

Algoritem SMGT so znanstveniki iz Univerze v Atenah predstavili kot rešitev vseh zgornjih problemov v scenariju QBH. Prednost njihove rešitve je v zmožnosti prilagajanja tolerance napak brez uporabe verjetnostnega modela. To je lahko dosegel tako, da dovoljuje vrzeli med poravnavo poizvedbenega ter ciljnega vzorca, omejuje največjo dolžino ciljnega vzorca in zahteva najmanjše število ujemajočih se elementov. To je bil prvi algoritem, ki je pri iskanju ujemanj dveh glasbenih posnetkov, upošteval vsa zgoraj navedena dejstva. To so podkrepili še z eksperimenti, s katerimi so dokazali, da je SMGT veliko boljši na področju natančnosti in učinkovitosti od vseh že obstoječih metod.

Z algoritmom SMBGT so Grški znanstveniki nadgradili svoj že odličen algoritem SMGT s še dvema dodatnima parametroma. Kot je razvidno že iz imena, so z *alfo* in *beto* omejili tudi število zaporednih vrzeli tako v poizvedbenem kot v ciljnem vzorcu. V zamrmranem poizvedbenem vzorcu lahko hitro nastanejo začasne napake v tempu ali intonaciji, kjer lahko manjka tudi kakšna cela nota. Ta parametra algoritmu omogočita preskok takšnih not.

Poglavje 4. Implementacija

4.1 Uporabljena orodja

V diplomski nalogi sta bili v večini uporabljeni dve razvojni orodji. Razvoj aplikacije, ki je služila kot pripomoček za lažje testiranje implementiranega sistema, je potekal v Android Studiu. Android Studio je uradno podprto razvojno okolje (angl. Integrated Development Environment ali pozneje IDE), ki se uporablja pri razvijanju aplikacij za platformo Android. Drugo orodje, Microsoft Visual Studio, se uporablja pri implementaciji DLL knjižnice, napisane v programskem jeziku C++, ter implementaciji iskalnih algoritmov v jeziku C#.

4.1.1 Android Studio

Android studio je uradni IDE za programiranje mobilnih aplikacij za naprave z operacijskim sistemom Android. Ker temelji na platformi IntelliJ IDEA[8], ima tudi podoben uporabniški vmesnik. Za namen uporabe pri razvoju Android aplikacije, ki je bila uporabljena pri preizkusih implementiranega sistema QBH, se je izkazalo kot nepogrešljivo orodje.

4.1.2 Microsoft Visual Studio

Microsoft Visual Studio je IDE, ki ga je razvil Microsoft. Primarno je uporabljen za razvoj aplikacij in programov za Microsoft Windows z manjšim poudarkom na spletnih aplikacijah, spletnih straneh ter drugih spletnih storitev. Zaradi uporabnosti pri razvijanju za strežnik Windows (angl. Windows Server) je bil v diplomski nalogi uporabljen kot primarno orodje. Za namen razvoja diplomskega dela je bila uporabljena storitev SVsSolution, ki je namenjena stvaritvi C# projektov in rešitev (angl. Solution).

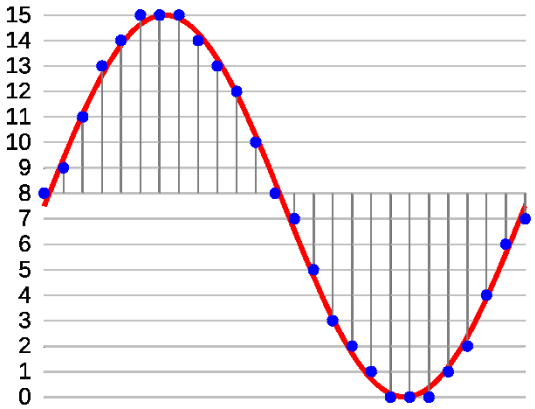
4.2 Android aplikacija

V razvojnem okolju Android Studio je bil narejen projekt, ki je bil poimenovan HummingApp. Za zajem zvoka s pomočjo mikrofona, vgrajenega v telefon, so bili poleg standardnih knjižnic za Androidne aplikacije potrebni še AudioRecord, AudioFormat in LinearLayout. Za shranjevanje avdio toka pa je bil uporabljen ByteBuffer.

Androidov programski vmesnik (angl. Application Programming Interface ali API) je zelo dobro dokumentiran, kjer se je nahajala tudi referenca na vmesnik AudioRecord. Vmesnik je namenjen

[8] IntelliJ IDEA je razvojno okolje, ki ga je razvilo podjetje jetBrains in je namenjeno programiranju programske opreme v programskem jeziku Java.

upravljanju z vsemi avdio viri za Androidno aplikacijo za zajem zvoka. To lahko doseže z branjem podatkov iz AudioRecord objekta, v katerega se shranjuje zvok kot velika tabela števil. Za uporabo vmesnika ga je treba kot pri vseh objektih, najprej inicializirati s pomočjo konstruktorja. V konstruktorju je treba podati število avdio kanalov, enkodiranje ter frekvenco vzorčenja. Uporabnik bo po navadi posnel relativno kratek del pesmi od 5 do 15 sekund, ki ne sme porabiti preveč prostora. Frekvenca vzorčenja je bila tako nastavljena na 22050 Hz. Enkodiranje je bilo standardno za takšno uporabo vmesnika, in sicer 16-bitno PCM[9]. Število kanalov je bil parameter, pri katerem ni imelo nobenega smisla izbrati več kot en kanal (mono).



Slika 12: Vzorčenje analognih signalov z metodo PCM.

Aplikacija je videti popolnoma preprosto. Vsebuje dva gumba, enega za začetek snemanja in drugega za pošiljanje posnetka ter prazen prostor, kamor se lahko vpiše ime posnetka. Slednji je bil dodan pozneje, saj je olajšal razlikovanje posnetkov v fazi preizkusov. Postavitev je bila definirana s pomočjo razreda LinearLayout.

Vsak posamezni posnetek je bil shranjen v tekstovno datoteko. V tej datoteki so bili zapisani snemalni parametri in tabela bajtov, ki so predstavljali posnete podatke. S strani algoritma za prepoznavanje osnovnih frekvenc je bilo najoptimalnejše, če so bili podani vhodni posnetki v takšni obliki. Aplikacija je služila kot pripomoček za lažje preizkušanje, zato niso bili v tej fazi vzpostavljeni še nobeni zaledni sistemi. Datoteke se je pošiljalo preko elektronskih sporočil, v katerih so se posnetki dodali kot priponke. V fazi integracije v končen sistem so se implementirali tudi zaledni sistemi, s pomočjo katerih uporabnik komunicira s podatkovno bazo.

4.3 Izvorna koda algoritma pYIN

Izvorna koda za algoritem pYIN, ki sta ga razvila Matthias Mauch in Simon Dixon z Univerze Queen Mary v Londonu, je na voljo na njuni spletni strani [9]. Spisana je v C++ jeziku in služi kot vtičnik

[9] *PCM ali "Pulse code modulation" je metoda, ki se uporablja za digitalno predstavitev vzorčenih analognih signalov. Je standardna oblika digitalnega avdia, ki se uporablja v mobilni telefoniji. Pri takšni predstavitvi avdio posnetka se amplituda analognega signala po navadi vzorči v enotnih intervalih, pri čemer je vsak vzorec kvantiziran na njemu najbližjo vrednost v določenem obsegu.*

za binarni modul Vamp, ki se uporablja pri analizi glasbe. Nekaj tipičnih stvari, za katere lahko uporabimo Vamp vtičnike, so:

- najdba lokacije začetkov not,
- izračunavanje moči zvoka v določenem času,
- vizualizacije avdio posnetkov, npr. spektrogrami,
- za določanje osnovne frekvence v izseku avdio posnetka.

Koda je odprtokodnega značaja, zato jo lahko uporabi in modificira kdor koli. V diplomskem delu niso bili potrebni vsi dodatki in funkcionalnosti, ki jih nudijo Vamp vtičniki, zato je bilo treba kodo prirediti. Vsak vtičnik nudi izpis kopice informacij o glasbenem posnetku, kot so npr. njegovo ime, unikatni identifikator, opis, avtorja, ki jih je bilo treba izluščiti, saj je bilo ustvarjeno posebno okolje kot nadomestilo za Vamp.

Ideja je bila, da se okrnjen Mauchov in Dixonov algoritem uporabi za procesiranje vhodnega avdio signala. Uporabnik lahko s pomočjo Android aplikacije zamrmra melodijo, ki jo potem procesira pYIN, ta ugotovi število, dolžino ter osnovne frekvence not. Ker je bil cilj prilagoditi cel postopek za delovanje na Windows serverju, je bilo potrebno kodo, spisano v C++ bodisi prepisati v C#, bodisi zapakirati v DLL knjižnico, ki se jo da z nekaj prilagoditvami uporabiti v C# kodi.

4.3.1 C++ opis

Programski jezik C++ se neposredno prevede v strojno (angl. native) kodo procesorja, kar ga naredi enega od najhitrejših programskih jezikov na svetu. Poleg hitrosti je tudi energijsko nezahteven. Dejstvo, da temelji na jeziku C, mu omogoča dobro kompatibilnost z večino C knjižnicami ter številnimi prevajalniki. Kot nenadzorovana (angl. unmanaged ali unsafe) koda, je C++ direktno nekompatibilna z jezikom C#, saj ima programer popoln nadzor nad dogajanjem v programu, vključno z nadzorom in upravljanjem s pomnilnikom ter zbiranjem smeti (angl. garbage collection). Jezik za dostop do pomnilnika uporablja kazalce, ki služijo kot referenca za objekt, nahajajoč se nekje v pomnilniku računalnika. V kazalcu se skriva naslov objekta, za katerega s pomočjo deklaracije razreda oz. podatkovnega tipa določimo število bajtov, ki jih bo zasedel v pomnilniku. Za vsak objekt z rezerviranim prostorom (angl. allocate) v pomnilniku moramo tudi poskrbeti, da se ob koncu uporabe s pomnilnika izbriše (angl. deallocate), kar spada pod nalogo upravljanja s spominom.

Za celotno kodo, napisano v jeziku C++, je bilo treba narediti nov razred (Handler.cpp), v katerem se nastavijo vsi parametri in služi kot vstopna točka. Da bi se izognili napačnemu upravljanju s pomnilnikom, je bil tako v glavni metodi razreda “Handler” uporabljen en kazalec, ki je označeval začetek tabele, pridobljene s posnetka v Android aplikaciji. Treba je bilo tudi podati velikost tabele. Za pravilno interpretacijo PCM tabele potrebuje algoritem frekvenco vzorčenja, pri kateri se je izkazalo, da je najoptimalnejša, če je enaka 22050 Hz. Posnetek s takšno frekvenco vzorčenja ne zasede preveč prostora, obenem pa še vedno vsebuje dovolj informacij za uporabo v pYIN algoritmu.

Algoritem kot parameter potrebuje tudi število avdio kanalov, ki ga je treba nastaviti na ena (mono). Ko ima algoritem podan kazalec na začetek PCM tabele ter vse druge potrebne parametre, začne procesirati podatke. Ta potek je natančneje opisan zgoraj.

Rezultat predstavlja dvodimenzionalna tabela not z njenimi lastnostmi - frekvenca, čas začetka in njeno trajanje, kar glavna funkcija razreda "Handler" tudi vrača. Za integracijo nenadzorovane (angl. unmanaged) kode v nadzorovano (angl. managed) okolje C# je bil razred "Handler" izvožen kot DLL knjižnica, ki je bila s pomočjo za to namenjenih orodij uvožena v glavni del programa, napisanega v C# jeziku.

4.3.2 Opis C++ .dll knjižnice

DLL je "Dynamic link library" in pomaga pri razvoju programov z modularno arhitekturo [27]. Knjižnica nima svoje main() metode, zato sama po sebi ne predstavlja delujočega programa. DLL je le modul, ki se lahko doda kompatibilnemu programu, s čimer se mu poveča funkcionalnost z vsemi v DLL-u vsebovanimi metodami. S pomočjo DLL knjižnice se lahko vsebovano kodo na preprost način uporabi na več mestih istega kot tudi v različnih programih hkrati. Z njo dosežemo učinkovitejšo uporabo s pomnilnikom in diskom, saj program zasede manj prostora. Vse to vpliva na hitrost zagona in delovanja tako operacijskega sistema kot programa.

Ustvarjanje nenadzorovane C++ knjižnice ni preveč zahtevno, če je koda napisana varno - programer uspešno poskrbi za zbiranje smeti, dobro upravlja spomin, ne pozabi na čiščenje spomina ipd. Poenostavljeno – podobno je ustvarjanju C++ projekta brez main() metode, ki se mu doda še nekaj ukazov.

1. Najprej se ustvari nov DLL projekt v Visual Studiu - *File* > *New* > *Project* > *Visual C++* > *Win32* > *Win32 Console application.*
2. Potem se specificira ime projekta ter ime rešitve ("solution").
3. Nato se v dialogu Win32 aplikacijskega čarovnika ("Application Wizard") izbere *Application Settings* > *Application Type* > *DLL.*
4. Potem se ustvari Header (.h) datoteko - *Project* > *Add New Item* > *Visual C++* > *Code* > *Header File.*
5. Dopolni se wrapper.h z ustrezno kodo.

```
#include <stdexcept>
using namespace std;

namespace wrapper
{
    extern "C" {__declspec(dllexport) double* CalculatePYIN(double* valPtr, int row, int col); }
    extern "C" {__declspec(dllexport) int ReleaseMemory(int* pArray); }
}
```

Slika 13: Koda Header datoteke razreda "wrapper".

6. Potem se doda vsa izvorna koda Mauchovega in Dixonovega algoritma ter koda razreda "Handler" in se jo dopolni z dvema zgoraj navedenima metodama.
7. DLL je treba le še prevesti s pomočjo prevajalnika - *Build > Build Solution.*

4.4 Prevajanje kode CPP v C# in njuna združljivost

4.4.1 C# opis

Programski jezik C# je objektno orientiran, preprost in modern. Razvil ga je Microsoft za svojo .NET platformo in je eden od številnih jezikov, ki so združljivi s skupno jezikovno infrastrukturo (angl. Common Language Infrastructure ali CLI). C# vsebuje vse dobre lastnosti dveh jezikov - C++ in Java. Tipi referenc so podobni kazalcem jezika C++, vendar programerju pri upravljanju s spominom pomaga prevajalnik, saj opozarja na vse napake v zvezi z ustvarjanjem in brisanjem objektov. Dobra lastnost jezika je tudi v tem, da zazna vse nekompatibilnosti tipov objektov, kar spet sporoči prevajalnik, ali pa aplikacija med svojim delovanjem sproži izjemo (angl. exception). Zaradi podpore modularnosti projektov lahko v .NET platformi kreiramo projekte, v katerih je združena koda več jezikov.

Za uporabo nenadzorovane C++ knjižnice v C# projektu je treba najprej dovoliti nadzorovani kodi klice nenadzorovanih metod, ki so implementirane v DLL z naslednjim ukazom.

```
using System.Runtime.InteropServices;
```

Slika 14: Izsek kode, ki omogoča uporaba klicev nenadzorovanih metod v nadzorovanem okolju.

Zgornjo vrstico je treba dodati na začetek programa. Za vsako deklarirano metodo v knjižnici, ki jo želimo uporabiti v glavnem programu, je treba navesti še pot do knjižnice ter navesti tip objekta, ki ga vrača, skupaj z vsemi njenimi argumenti.

```
[DllImport("C:\\Users\\tadejmittoni\\ConsoleApplication1.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern IntPtr CalculatePYIN(IntPtr valPtr, int row, int col);
[DllImport("C:\\Users\\tadejmittoni\ConsoleApplication1.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern int ReleaseMemory(IntPtr ptr);
```

Slika 15: Izsek kode, v katerem je navedena pot do knjižnice, ki vsebuje želene metode, vključno s tipi objekta, ki ga vsaka posamezna metoda vrača.

Za lažje in bolj urejeno upravljanje z zgornjima dvema metodama je bila vsa potrebna koda zapakirana v naslednjo metodo.

```
private static double[] pyinStuff(double[,] entry)
{
    double[] cppArray = new double[entry.GetLength(0) * entry.GetLength(1)];
    int cppArrayPoint = 0;
    for (int i = 0; i < entry.GetLength(0); i++)
    {
        for (int j = 0; j < entry.GetLength(1); j++)
        {
            cppArray[cppArrayPoint] = entry[i, j];
            cppArrayPoint++;
        }
    }

    GCHandle handle = GCHandle.Alloc(cppArray, GCHandleType.Pinned);
    var entrPtr = handle.AddrOfPinnedObject();
    int row = entry.GetLength(0);
    int col = entry.GetLength(1);


    IntPtr ptr = neki(entrPtr, row, col);
    double[] velikost = new double[1]; //7
    Marshal.Copy(ptr, velikost, 0, 1);
    double[] rez = new double[(int)velikost[0]];
    Marshal.Copy(ptr, rez, 0, (int)velikost[0]);

    ReleaseMemory(ptr);

    return rez;
}
```

Slika 16: Izsek kode metode pyinStuff() v C#.

Zgornja slika predstavlja metodo, ki uporablja metode iz .DLL knjižnice. Služi za komunikacijo med jeziki, saj je treba na prehodu v »unmanaged« kodo ročno upravljati s spominom. Za alokacijo spomina se uporablja »GCHandle« z metodo AddrOfPinnedObject(), ki nam vrne kazalec na objekt, alociran v spominu s strani C++ kode. Prva vrednost v tabeli predstavlja velikost, v preostanku tabele pa so shranjene informacije o zamrmranem vzorcu, ki smo jih pridobili s pomočjo algoritma pYIN.

4.5 Opis implementacije baze podatkov na strežniški strani

Pri implementaciji baze podatkov je bila uporabljena knjižnica NAudio, ki je odprtokodnega značaja in je napisana za programski jezik C#. Uporabniku omogoča branje različnih tipov avdio datotek in še veliko več. Z njeno pomočjo se najprej prebere vse MIDI datoteke iz mape na disku v seznamu. Seznam, ki predstavlja vse posnetke iz baze, se med eksperimentom ves čas obdrži v spominu, kar omogoča hiter dostop. Podatkovno bazo v spominu predstavlja objekt »Database«, v katerem so vse veljavne MIDI datoteke, ki jih predstavlja objekt »Song«. Vsak objekt »Song« vsebuje podatke o pesmi, in sicer njeno ime, identifikacijsko število ter zaporedje not. Vsako noto predstavlja objekt »Note«, ki o njej hrani podatke o dolžini, času trajanja ter MIDI številko višine note. Te številke se začnejo pri 21, ki predstavlja noto A0 s frekvenco 27.5 Hz. Ko so vse note pesmi prebrane v objekt

»Song«, se nad njim izvrši operacija *calculateAllEncodings()*, ki iz not izračuna vseh osem možnih predstavitev trenutne pesmi. Vsaka posamezna predstavitev pesmi je natančneje opisana v poglavju o algoritmu SMBGT. Vsako posamezno pesem se za tem doda v podatkovno bazo »db«. Podatki o vseh pesmih v podatkovni bazi se ob vstavljeni zadnji pesmi pripravijo za uporabo v iskalnih algoritmih. S klicem *prepareDbPitchAndDbRatioArrays()* nad bazo se združi vse izračunane tabele predstavljene pesmi v eno skupno. Ko podatkovna baza vsebuje osem tabel s podatki o zaporednih notah vseh pesmi, je pripravljena za poznejšo uporabo v iskalnih algoritmih.

```
String folderPath = "C:\\Users\\tadejmittoni\\Downloads\\EtnoFletno"; //database path
Database db = new Database(folderPath);
int songId = 0;

foreach (string file in Directory.EnumerateFiles(folderPath, "*.mid"))
{
    String fileName = Path.GetFileName(file);
    MidiFile currMidi = new MidiFile(file, false);
    Song currSong = new Song(fileName, songId);

    foreach (MidiEvent note in currMidi.Events[1])
    {
        if (note.CommandCode == MidiCommandCode.NoteOn)
        {

            var t_note = (NoteOnEvent)note;
            String name = t_note.NoteName;
            int number = t_note.NoteNumber;
            int length = t_note.NoteLength;
            long absTime = t_note.AbsoluteTime;

            Note currentNote = new Note(name, number, length, absTime);
            currSong.AddNote(currentNote);

        }
    }
    currSong.calculateAllEncodings();
    db.addSongToList(currSong);
    songId++;
}
db.prepareDbPitchAndDbRatioArrays();
```

Slika 17: Izsek kode, ki se uporablja pri vzpostavitvi podatkovne baze.

Vhodni parametri algoritma SMBGT, ki so bili povezani s podatkovno bazo, so bili:

- enodimenzionalna tabela višin not vseh pesmi v podatkovni bazi,
- enodimenzionalna tabela IOIR vseh pesmi v podatkovni bazi,
- število vseh pesmi v podatkovni bazi,
- enodimenzionalna tabela števila not posamezne zaporedne pesmi v podatkovni bazi.

Prva dva parametra sta bila prilagojena glede na trenutno izbran način predstavitve pesmi, druge dva pa sta algoritmu služila za pomoč pri krmarjenju skozi tabele predstavitve podatkovne baze. Časovna zahtevnost vzpostavitve baze je majhna, saj se mora v tem delu program le dvakrat sprehoditi skozi vse note pesmi.

4.6 Opis implementacije vhodnega signala na strežniški strani

Zgoraj omenjena metoda *pyinStuff()* je kot vhodni parameter dobila tabelo bajtov, ki jo je s pomočjo algoritma pYIN spremenila v zaporedje not. Zaporedje not je vsebovalo podatke o frekvenci, časovnem žigu ter dolžini vsake posamezne note. Vhodni signal je bilo potem treba shraniti v objekt tipa »Song«, pri čemer je pomagal spodnji izsek kode.

```
double[] query = pyinStuff();

Song querySong = new Song("query", -1);
for (int i = 1; i < query.Length; i+=3) {
    double fm = query[i];
    int number = (int)(Math.Round(12*Math.Log(fm/440, 2) + 69));
    long absoluteTime = (int)query[i + 1];
    int length = (int)query[i + 2];

   Note currNote = new Note(number, length, absoluteTime);
   querySong.AddNote(currNote);
}
querySong.calculateAllEncodings();
```

Slika 18: Izsek kode, ki je pomagala pri pretvorbi podatkov pridobljenih s pomočjo algoritma pYIN v objekt tipa "Song".

V tabeli, ki jo vrača metoda *pyinStuff()*, so bile zaporedne note predstavljene v eni dimenziji. Vsako noto so torej predstavljale tri zaporedne številke, od katerih je bila prva frekvenca, druga časovni žig ter tretja dolžina note. Enačba $Math.Round(12 * Math.Log(fm/440, 2) + 69)$ se je uporabljala pri pretvorbi iz frekvence v MIDI predstavitev note, pri čemer fm predstavlja frekvenco note. Za *querySong* kot za vse druge objekte tipa »Song« je bilo treba pred uporabo v iskalnih algoritmih izračunati vseh osem predstavitev s pomočjo metode *calculateAllEncodings()*.

Poglavje 5. Poskusi in rezultati

V raziskavi znanstvenikov z Univerze v Atenah je precejšen del posvečen preizkušanju vseh omenjenih algoritmov iskanja. Svoje domneve so podkrepili še z eksperimentalnim delom. Zaradi pomanjkanja prilagodljivosti do zdaj poznanih algoritmov na temo iskanja podobnosti med dvema glasbenima vzorcema se je izkazalo, da niso učinkoviti. S sintetičnimi glasbenimi vzorci, ki so jim dodali različno količino šuma, so dokazali, da se učinkovitost algoritmov drastično zmanjša s povečevanjem količine šuma v vzorcu, zato se jih pri iskanju z zamrmranimi vzorci ne more uporabiti.

Od vseh algoritmov sta se najbolje obnesla SMBGT ter SMGT, ki s številom nastavljivih parametrov ponujata veliko fleksibilnosti pri vhodnih vzorcih. Od že poznanih algoritmov je bil najobetavnejši »Edit distance«, ki pa je vseeno bil tridesetkrat manj uspešen (angl. recall rate) pri zamrmranih vzorcih za K = 50 (K predstavlja vnaprej določeno število želenih rezultatov) od algoritmov SMBGT ter SMGT. Pri manjšem številu K je bil SMBGT za 15 % boljši od SMGT, kar potrdi domnevo, da je vpeljava omejitev *alfe* in *bete* smiselna. Prav tako se potrdi domneva, da je konstantna toleranca vedno slabša od spremenljive. *Alfa* in *beta* sta bili prilagojeni glede na smiselne kombinacije za zamrmrane vzorce, in sicer na cela števila, ki se nahajajo na intervalu [4, 6]. *Delta,* ki predstavlja decimalno število tolerance, je bila prilagojena glede na pevske sposobnosti osebe, ki je zamrmrala določen vzorec. Kadar uporabnik posname dober vzorec (ujame pravilno višino ter tempo), se *delto* nastavi na *0.5* do največ *0.5 * dolžina vzorca*. Parameter *r* (»range« – največja dolžina ujemajočega vzorca) naj ne bi presegal dolžine zamrmranega vzorca s faktorjem 1,2. Glede vrst predstavitev se je izkazalo, da so preprostejše najučinkovitejše pri vseh algoritmih, ki vsebujejo metode dinamičnega programiranja. Prav tako imajo takšne metode majhno časovno zahtevnost. Algoritmi, ki uporabljajo metode SMM, so sicer hitrejši, vendar je njihov učni postopek računsko zahtevnejši.

5.1 Testno okolje

Približen obseg vrednosti, ki jih preostali parametri algoritmov SMGT ter SMBGT zajemajo, so bili poznani, vseeno pa je bilo okolje postavljeno tako, da je kar se da fleksibilno. Sledila je optimizacija vseh parametrov, ki vplivajo na natančnost rezultatov algoritma:

- ***delta***, najmanjše število ujemajočih se elementov,
- ***maxr***, omejitev dolžine ujemajočega se vzorca,
- ***alpha***, število vrzeli v ciljnem vzorcu,
- ***beta***, število vrzeli v poizvedbenem vzorcu,
- ***vrsta tolerance***, absolutna ali relativna,
- ***spremenljivost tolerance***, spremenljiva ali konstantna,

- ***vrednost tolerance (tol_value)***, toleranca algoritma do napak v posnetem vzorcu,
- tip predstavitve.

```
for (int Trepr = TreprStart; Trepr < TreprCond; Trepr += TreprStep)
{
    for (double Td = TdStart; Td < TdCond; Td += TdStep)
    {
        for (double Tmaxr = TmaxrStart; Tmaxr < TmaxrCond; Tmaxr += TmaxrStep)
        {
            for (int Talpha = TalphaStart; Talpha < TalphaCond; Talpha += TalphaStep)
            {
                for (int Tbeta = TbetaStart; Tbeta < TbetaCond; Tbeta += TbetaStep)
                {
                    for (double Ttol_val = Ttol_valStart; Ttol_val < Ttol_valCond; Ttol_val += Ttol_valStep)
                    {
```

Slika 19: Glavne zanke testnega okolja algoritma SMBGT.

Programski del preizkusov je predstavljala metoda, v kateri se je nastavljajo vse želene parametre. Vsebovala je šest *for* zank, v katerih se je iteriralo čez vse možne vrednosti parametrov, določenih z intervali na začetku metode, ki so prikazane na Sliki 18. Omejeno je bilo tudi število rezultatov, in sicer najprej na 15 (zgoraj omenjena vrednost *K),* pri večji bazi na 40. Inicializirana sta bila tudi dva seznama rezultatov. Seznama sta delila rezultate na neuspešne ter uspešne. V glavnem delu analize rezultatov je bil algoritem SMBGT zagnan z vsako kombinacijo parametrov. Iz tega je bila pridobljena tabela ocen razdalj zamrmranega vzorca do najbolj podobnih *K* vzorcev iz baze. Če je bil želeni ciljni vzorec med njimi, je bil rezultat dodan v primeren seznam.

Vsak rezultat preizkusa je vseboval informacije o parametrih, s pomočjo katerih je bil dosežen, zaporedno številko iteracije ter seznam K najboljših vzorcev. S seznama najboljših vzorcev je bilo pridobljenih še več informacij, ki so pomagale pri končni ocenitvi trenutnih parametrov:

1. oceno razdalje zamrmranega vzorca do vsakega posameznega vzorca izmed K najboljših,
    - najboljša ocena je bila 10, najslabša 2, manj kot 2 sem tretiral kot neuspešen poizkus;
2. mesto ciljnega vzorca med najboljšimi K vzorci;
3. razdalja do drugega vzorca,
    - če je bil želeni vzorec na prvem mestu,
    - večja, kot je bila, boljši je bil poizkus.

Na uspešnost kombinacije parametrov iteracije je najbolj vplivalo mesto ciljnega vzorca med najboljšimi K vzorci, saj ta parameter najbolj vpliva na uporabnikovo izkušnjo ob uporabi aplikacije. Idealen algoritem bi vedno našel ciljni vzorec na prvem mestu, ki si ga ne bi delil z nobenim drugim vzorcem. Takšen rezultat bi bilo seveda pri vseh faktorjih, ki jih algoritem upošteva, mogoče dobiti le, če bi uporabnik zamrmral vzorec s popolno natančnostjo in bi bili parametri algoritma tako tudi nastavljeni. V praksi je nemogoče doseči popolno okolje, saj bi bilo treba v celoti izničiti šum okolice. Natančnost posnetka vhodnega signala se najprej zmanjša z vzorčenjem, na katerega vpliva tudi kakovost mikrofona v mobilni napravi, potem pa še pri ugotavljanju osnovne frekvence s pomočjo algoritma pYIN, ki je uspešen v 95 % primerih. Upoštevati je seveda treba tudi pevske sposobnosti osebe, ki zamrmra vhodni signal.

Končna ocena kombinacije parametrov je bila dodana v ožji izbor v primeru, da je bil iskani vzorec vsaj med K najboljšimi ujemajočimi se vzorci. V nasprotnem primeru se je kombinacija štela kot neuspešna. Kombinacije parametrov, pri katerih je bil iskani vzorec med petimi najboljšimi oz. si je delil vsaj drugo mesto, je bila še dodatno označena. Vsi izvedeni poizkusi so se glede na njihovo uspešnost ob koncu shranili tudi v primerne sezname.

Razred rezultatov poizkusov vsebuje tudi metodo izpisa v datoteko, ki iterira nad vse dobljene rezultate ter zapiše zgoraj omenjene parametre v tekstovno datoteko. Izpis vseh K rezultatov je bil še dodatno omejen tako, da se je končal, ko je bil za tisto kombinacijo izpisan ciljni vzorec.

5.2 Preizkušanje sistema s pomočjo pevcev

5.2.1 Opis poizkusa in baze podatkov

Glede na vse že opravljene raziskave, ki so opisane na začetku četrtega poglavja, je bilo dobro razvidno, kako je treba izvesti preizkušanje učinkovitosti implementacij algoritmov na naši bazi podatkov. Ta je bila namreč sestavljena iz starih slovenskih pesmi, ki so dostopne na www.etnofletno.si, kjer se nahaja tudi implementacija sistema QBH omenjenega v diplomski nalogi. Podatkovna baza je na začetku preizkušanja vsebovala 61 MIDI datotek, pozneje nekaj več kot 600.

Preizkušane so bile vrednosti parametrov v intervalih, in sicer:

- *delta = [0, 0.7] s korakom 0.05,*
- *beta = [0, 7] s korakom 1,*
- *alfa = [0, 7] s korakom 1,*
- *maxr = [0.9, 1.5] s korakom 0.05,*
- *tol_value = [0, 0.25] s korakom 0.05,*
- *tol_type = [1, 2], pri čemer je 1 absolutna in 2 relativna,*
- *tol_variability = [1, 2], pri čemer je 1 spremenljiva in 2 konstantna,*
- *repr_type = [0, 7], pri čemer vsaka številka predstavlja predstavitev iz spodnje tabele.*

ID	Predstavitev višine	Predstavitev časa
0	Mod12	IOIR
1	Mod12	LogIOIR
2	Mod12	LogIOIR [-2, 2]
3	Mod12	Rounded LogIOIR
4	Višinski interval	IOIR
5	Višinski interval	LogIOIR
6	Višinski interval	LogIOIR [-2, 2]
7	Višinski interval	Rounded LogIOIR

Najprej je bila preizkušena postavitev sistema, tako da je bil kot zamrmrani vzorec vzet kar naključen vzorec iz podatkovne baze. Primerno so bili nastavljeni vsi parametri, in sicer tako, da niso dopuščali nobenih napak. Uspešnost algoritmov je bila 100%, kar je bilo glede na ustreznost parametrov pričakovano. S tem je bilo dokazano, da v popolnem scenariju algoritmi delujejo brezhibno. Realne scenarije sem preizkusil s pomočjo zelo in malo manj šolanih pevcev. Ciljne datoteke so bile štiri. Vsak pevec je po nekajkratnem poslušanju poskusil štirikrat kar se da dobro zamrmrati pesem. Preizkušeno je bilo tudi petje "nana", ki pa se je izkazalo za manj uspešno. Preizkusi so bili tako izvedeni s 96 različnimi vzorci.

5.2.2 Sposobnosti pevcev, opis prostora in potek poizkusa

Pri preizkušanju sistema je pomagalo šest pevcev. Dva sta šolana, dva navdušenca za petje in dva, ki se s petjem ne ukvarjata. Oba šolana pevca sta več let obiskovala glasbeno šolo, navdušenca za petje pa pojeta v zboru že več let, vendar nikoli nista obiskovala glasbene šole. V vsakem paru je bila ena oseba ženskega in ena moškega spola. Vsem pevcem so bile ciljne datoteke, razen pesmi Yesterday skupine The Beatles, neznane, prav tako še niso slišali za noben uporabljen algoritem. Snemanje posnetkov se je odvijalo v zaprtem prostoru, ki je bil zvočno dobro izoliran.

Vsakemu paru je bilo najprej v grobem razložen potek testiranja, saj tako njihovo poznavanje algoritma ni vplivalo na rezultate. Za vsako melodijo so bili posneti štirje vzorci na osebo. Najprej kratek (do šest not) in potem dolg (več kot 10 not) ter vse ponovljeno skupaj dvakrat. Osebi iz prvega in drugega para sta po štiri- do šestkratnem poslušanju pesmi posneli prvi par vzorcev, tretji par jo je poslušal osemkrat. Po posnetem prvem paru vzorcev je bilo vsem v grobem razloženo delovanje algoritma, na kaj je najbolj občutljiv ter na kakšne napake naj bodo najbolj pozorni. Začetna intonacija ne vpliva na rezultate, dokler premiki med višinami not ostanejo nespremenjeni, prav tako je treba bolj paziti na ujemanje tempa ter glasnost vmesnih vdihov. Z vsem tem znanjem so pevci posneli še preostale. Če je pevec ali pevka med snemanjem določenega vzorca naredil kakšno očitno napako, je bil vzorec izbrisan. Pesem Yesterday je bila obravnavana kot zadnji primer.

5.2.3 Analiza rezultatov in ugotovitve

Odlična ocena kombinacije parametrov je bila v primeru, da je algoritem našel iskani vzorec med petimi najboljšimi, ali pa si je delil vsaj drugo mesto. Dobro oceno je dobila kombinacija parametrov, katerih ciljni vzorec je bil v K najboljših ujemajočih se vzorcev. Sisteme QBH se po navadi sicer ocenjuje z veliko bazo, več kot 2000 MIDI datotek, za uspešen rezultat pa se šteje primere, ko sistem vrne pravilno pesem med K = 10 najbolj podobnih iz baze. Takšno ocenjevanje bi bilo smiselno, če bi sistem QBH, ki je opisan v diplomskem delu, vseboval bazo glasbenih del raznovrstnih žanrov in ne le enega. V primeru omenjenega sistema je bil glavni cilj optimizacija parametrov glede na

podatkovno bazo le slovenskih ljudskih pesmi, bi pa bilo smiselno v prihodnosti uspešnost tega sistema QBH preizkusiti tudi glede na druge klasične sisteme, kot to delajo z oceno MIREX[10].

Pri majhni bazi (61 pesmi) je algoritem deloval dobro. Za več kot 25 % zamrmranih vzorcev prvega ali drugega para oseb je obstajala vsaj ena kombinacija parametrov, ki so vodili do odlične ocene. Najboljšo oceno je dosegel fant iz prvega para, pri katerem je bil ciljni vzorec na prvem mestu s še štirimi drugimi vzorci. K temu je prispevalo dejstvo, da je imelo vseh pet precej podobno melodijo oz. razmerja med višinskimi preskoki not. Dobro oceno je doseglo 63 % primerov, dva vzorca pa za K = 15 nista bila uspešna. Poleg obeh kratkih posnetkov, sta bila neuspešna tudi oba posnetka prvih poizkusov. Kratki vzorci so se tako izkazali za veliko slabše od daljših, verjetno zaradi dejstva, da so bile razlike med višinami sosednjih not pesmi v bazi med seboj precej podobne. Uspešnost tretjega para je bila obravnavana posebej, saj nista del ciljnega občinstva. Odrezala sta se slabše od drugih, saj je odlično oceno le enkrat doseglo dekle, katere ciljni vzorec je bil na drugem mestu s še tremi drugimi vzorci. Ostale ocene tretjega para so bile v 80 % primerov neuspešne, saj med K = 15 ciljnega vzorca ni bilo. Dva vzorca sta bila ocenjena dobro, vendar s popolnoma drugačnimi parametri kot pri uspešnih poizkusih prvih dveh parov.

Algoritem je na veliki bazi (612 pesmi) deloval slabše. Za K = 15 so bili vsi vzorci vsaj za oceno slabši, kar pomeni, da so vzorci z odlično oceno dosegli najboljšo oceno dobro, tisti z dobro oceno pa se med K najboljšimi niso več nahajali. En odličen vzorec iz majhne je bil v veliki bazi neuspešen. Takšni rezultati so bili pričakovani, saj je tukaj še bolj do izraza prišla medsebojna podobnost pesmi v bazi. Pri K = 40 so bili rezultati boljši in bolj podobni rezultatom v majhni bazi, razen enega odličnega vzorca, v kateri mi je padla ocena na dobro. Odličen rezultat vzorca mrmranja neznane melodije v bazi zelo podobnih pesmi se je tako izkazal za precejšen izziv.

Vsi so se najbolje odrezali pri mrmranju pesmi Yesterday. To je bila edina pesem v majhni bazi, ki ni bila slovenska ljudska. Pevčevo predhodno poznavanje pesmi je v tem primeru najbolj vplivalo na tako dober rezultat. Pri vzorcih petja slovenskih ljudskih pesmi je najboljšo (odlično) oceno doseglo dekle iz prvega para, kjer je bil ciljni vzorec sam na prvem mestu. Odlično oceno je dosegel tudi vzorec, ki sem ga zapel sam, vendar šele v šestem poizkusu. Oba pevca iz tretjega para sta z dolgim vzorcem dosegla dobro oceno.

Kot je bilo že dokazano v raziskavah, ki so podpirale razvoj algoritma SMBGT, se izkazalo, da je absolutna toleranca veliko boljša od relativne. Prav tako se je najpreprostejša predstavitev vzorca ter podatkovne baze izkazala za najučinkovitejšo. Uporabljena je bila naslednja predstavitev: <višinski interval, IOIR (razlika v času med začetkoma dveh sosednjih not)>. Eksperimenti so tudi podprli že dokazano doseženo boljšo učinkovitost algoritma pri spremenljivi toleranci. Ostali parametri so se

[10] MIREX ali »Music Information Retrieval Evaluation eXchange« [MIREX] uporablja ocenjevanje z vnaprej definirano bazo tako glasbenih del kot tudi poizvedb. QBH sistem se ocenjuje tako, da se mu za vsako poizvedbo, kjer sistem najde pravilno glasbeno delo iz podatkovne baze med najbolj podobnimi desetimi deli, doda ena točka, v nasprotnem primeru ne dobi nič točk. Velja kot standard za ocenjevanje klasičnih QBH sistemov.

razlikovali glede na pevske sposobnosti osebe, ki mrmra. V raziskavi so določili predlagane vrednosti parametrov sledeče: *delta = 0.5, beta = 6, alfa = 5, maxr = 1,2* in *tol_value = 0,2*. Med eksperimentiranjem se je izkazalo, da se optimalni parametri rahlo razlikujejo od predlaganih. Upoštevano je bilo namreč, da je ciljno občinstvo uporabe opisane implementacije algoritma bolj izobraženo na področju glasbe in bo imelo vsaj takšno znanje, kot sta ga imeli osebi iz drugega para pevcev. Precejšnja razlika je bila tudi v lastnostih pesmi v podatkovni bazi. V primeru diplomske naloge je bila celotna baza sestavljena iz pesmi istega žanra. Slednje dejstvo ni dovoljevalo prevelikega popuščanja pri toleranci algoritma do napak, saj je bilo drugače dobljeno preveliko število pesmi, ki so si delile prvo ali drugo mesto. Prvi par je dosegel najboljše rezultate pri *delta = 0.15, beta = 2, alfa = 1, maxr = 1,05* in *tol_value = 0,15,* medtem ko so bili najoptimalnejši parametri pri drugem paru: *delta = 0.2, beta = 3, alfa = 2, maxr = 1,1* in *tol_value = 0,2*.

5.3 Zaključek

V tem diplomskem delu je bilo pokazano, da najoptimalnejši sistem QBH za uporabo v spletni aplikaciji EtnoFletno sestavlja algoritem za transkripcijo zamrmranega vzorca pYIN ter algoritem za iskanje zamrmranemu vzorcu najbolj podobnih v podatkovni bazi SMBGT. Poleg dokaza je bil takšen sistem tudi implementiran. Končni sklep iz dobljenih rezultatov je, da je pri uporabi implementacije najbolj smiselno narediti dva sklopa parametrov algoritma SMBGT, ki si jih bo vsak uporabnik lahko nastavil sam. V izogib zmedi bi lahko bila poimenovana le »Sem odličen pevec« in »Sem dober pevec«. Tako bi algoritem ostal do neke mere nastavljiv in bi s tem vračal boljše rezultate, hkrati pa bi za uporabnika izkušnja ostala kar se da preprosta. Baza je velika in vsebuje pesmi, ki so si med seboj precej podobne. Zaradi tega doseganje odličnih rezultatov žal ni mogoče, lahko pa dobremu pevcu vseeno pomaga zožiti izbor pesmi iz velike baze na veliko manj. To lahko s pomočjo algoritma doseže le z mrmranjem melodije, kar je preprosto in hitro.

Če bi se pojavila želja po izboljšavi trenutnega sistema, se zdi najbolj smiseln naslednji korak implementacija algoritma, ki bi se prilagodil vsakemu posameznemu uporabniku posebej. To se lahko doseže z učenjem, kar pa seveda spremeni uporabnikovo izkušnjo. Ideja je takšna, da bi vsak uporabnik na začetku dobil nekaj vnaprej določenih glasbenih posnetkov, ki bi jih moral zamrmrati, potem bi algoritem z začetnimi parametri vrnil nekaj najbolj podobnih posnetkov iz podatkovne baze. Vsak uporabnik bi nato ocenil natančnost algoritma za vsak posamezni učni vzorec, kar bi ponovil vsaj nekajkrat. Algoritem bi si na tak način lahko izdelal profil uporabnika, saj bi vsak parameter glede na oceno natančnosti rezultata spreminjal, kar bi omogočilo algoritmu prilagajanje na unikatne napake vsakega posameznega uporabnika. Morda ima uporabnik slabši smisel za ritem, spet drugi ima napačno začetno intonacijo ali pa celo potrebuje več vmesnih vdihov, na kar bi se algoritem odzval s povečanjem parametra *alfa* ali *beta*. Prilagajali bi se tudi vsi drugi parametri, kar bi izboljšalo natančnost sistema QBH. Z implementacijo učenja bi vsak uporabnik v iskanje moral vložiti več časa,

vsaj dokler je algoritem še v stanju profiliranja, bi pa bil zato rezultat boljši. S tem bi se spremenilo tudi ciljno občinstvo iz nezahtevnega v zahtevnejšega uporabnika.

Za zdaj univerzalni QBH sistem, ki bi bil tako učinkovit na vseh glasbenih področjih, kot tudi zelo preprost za uporabnika, še ne obstaja. To dejstvo vodi do razvoja različnih sistemov QBH in vedno večje želje po odkritju univerzalnega, ki bo združil vse dobre lastnosti obstoječih. Glede na hitrost razvoja tehnologije na tem področju ter razvoj glasbene industrije, ki ga diktira človeštvo, je verjetno le vprašanje časa, kdaj bo nekomu to tudi uspelo.

Literatura

[1] D. R. M. M. R. Y. M. Alexandros Nanopoulos, „Music search engines: Specifications and challenges," *Information Processing and Management,* p. 392–396, 2009.

[2] D. R. B. P. David Little, „Online Training of a Music Search Engine," Electrical Engineering and Computer Science Department, July 12th, 2006.

[3] D. W. K.P.P.S. Warnaweera, „QBH System Using Melody Matching Model based on a String Pattern Matching mechanism of Frequency Contours," *International Journal of Emerging Technology and Advanced Engineering,* pp. 372-375, September 2012.

[4] S. Hjelmqvist, „CodeProject," 26 3 2012. [Elektronski]. Available: http://www.codeproject.com/Articles/13525/Fast-memory-efficient-Levenshtein-algorithm.

[5] J. L. W.-Q. Z. Jingzhou Yang, „A Fast Query by Humming System Based on Notes," *Tsinghua National Laboratory for Information Science and Technology,* pp. 2898-2901, 26-30 September 2010.

[6] P. P. J. H. D. G. V. A. G. K. Alexios Kotsifakos, „Hum-a-song: A Subsequence Matching with Gaps-Range-Tolerances Query-By-Humming System," *Proceedings of the VLDB Endowment, Vol. 5, No. 12,* pp. 1930-1933, 2012.

[7] H. C. Z. W. Yingmin Li, „Dynamic Time Warping Distance Method for Similarity Test of Multipoint Ground Motion Field," *Mathematical Problems in Engineering,* p. 12, 2010.

[8] H. K. Alain de Cheveigne, „YIN, a fundamental frequency estimator for speech and music," *Acoustical Society of America,* pp. 1917-1930, 2002.

[9] S. D. Matthias Mauch, „pYIN: A Fundamental Frequency Estimator Using Probabilistic Threshold Distributions," v *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2014)*, London, 2014.

[10] M. M. a. C. C. a. R. B. a. G. F. a. J. S. a. J. D. a. J. B. a. S. Dixon, „Computer-aided Melody Note Transcription Using the Tony Software: Accuracy and Efficiency," v *Proceedings of the First International Conference on Technologies for Music Notation and Representation*, 2015.

[11] D. Gerhard, „Pitch Extraction and Fundamental Frequency: History and Current Techniques," Technical Report TR-CS 2003-06, Regina, Saskatchewan, 2003.

[12] S. A. Z. Kavita Kasi, „YET ANOTHER ALGORITHM FOR PITCH TRACKING," Department of Electrical and Computer Engineering, Norfolk, USA.

[13] D. Talkin, „A Robust Algorithm For Pitch Tracking," *Speech Coding and Synthesis,* pp. 495-518, 1995.

[14] W. Hess, „Pitch Determination of Speech Signals," Springer-Verlag, Berlin, 1983.

[15] J. C. R. Licklider, „A duplex theory of pitch perception," *Experientia 7,* p. 128–134, 1951.

[16] B. C. J. Moore, „An Introduction to the Psychology of Hearing," Academic, London, 1997.

[17] R. a. H. M. J. Meddis, „Virtual pitch and phase sensitivity of a computer model of the auditory periphery. I: Pitch identification," *J. Acoust. Soc. Am. 89,* p. 2866–2882, 1991.

[18] P. A. a. D. B. Cariani, „Neural correlates of the pitch of complex tones. I. Pitch and pitch salience," *J. Neurophysiol. 76,* p. 1698–1716, 1996.

[19] J. L. Goldstein, „An optimum processor theory for the central formation of the pitch of complex tones," *J. Acoust. Soc. Am. 54,* pp. 1496-1516, 1973.

[20] F. L. Wightman, „The pattern-transformation model of pitch," *J. Acoust. Soc. Am. 54,* p. 407–416, 1973.

[21] E. Terhardt, „Pitch, consonance and harmony," *J. Acoust. Soc. Am. 55,* p. 1061–1069, 1974.

[22] S. D. D. G. H. K. A. K. Emmanouil Benetos, „Automatic Music Transcription: Breaking The Glass Ceiling," *Centre for Digital Music, Queen Mary University of London,* pp. 379-384, 2012.

[23] M. Blondel, „Mathieu's log," 2009. [Elektronski]. Available: http://www.mblondel.org/journal/2009/08/31/dynamic-time-warping-theory/.

[24] P. R. H. S. Christopher D. Manning, Introduction to Information Retrieval, Stanford: Cambridge University Press, 2008.

[25] C. F. M. Y. Yasushi Sakurai, Stream Monitoring under the Time Warping Distance, NTT Cyber Space Laboratories, Carnegie Mellon University, 2007.

[26] P. P. J. H. D. G. Alexios Kotsifakos, „A Subsequence Matching with Gaps-Range-Tolerances Framework: A Query-By-Humming Application," *Proceedings of the VLDB Endowment,* pp. 761-771, 2011.

[27] Microsoft, „Microsoft Support," 2016. [Elektronski]. Available: https://support.microsoft.com/en-us/kb/815065.

[28] H. H. Stephen A. Zahorian, „A spectral/temporal method for robust fundamental frequency," *Acoustical Society of America,* p. 4559–4571, 2008.

[29] H. a. C. S. Sakoe, „Dynamic programming algorithm optimization for spoken word recognition," *IEEE Transactions on Acoustics, Speech and Signal Processing,* pp. 43-49, 1978.

[30] T. Grandke, „Interpolation Algorithms for Discrete Fourier Transforms of Weighted," *IEEE Trans. Instrumentation and Measurement, Vol. IM-32,* pp. 350-355, 1983.

[31] M. S. P. Judith C. Brown, „A high resolution fundamental frequency determination based on phase changes of the Fourier transform," *J. Acoust. Soc. Am. 94,* pp. 662-667, 1993.

[32] IMIRSEL, „International Music Information Retrieval Systems Evaluation Laboratory," 2010-2015. [Elektronski]. Available: http://www.music-ir.org/mirex/wiki/2015:Query_by_Singing/Humming.